

Virtuoso Technology Data Constraints Reference

**Product Version IC23.1
November 2023**

© 2023 Cadence Design Systems, Inc.
Printed in the United States of America.

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

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

Contents

1

Constraint Groups and Constraints 17

Constraint Groups 18

constraintGroups 19

Constraint Group Definitions 21

User-Defined Constraint Groups 24

Combining Constraints in a Constraint Group 25

Constraints 28

Spacing Table Constraints 30

Constraint Parameters 32

'hard 32

'soft 32

'coincidentAllowed 33

'description 33

'ref 34

'manhattan 34

Advanced Nodes Constraints 35

Supported Constraints and Parameters: Virtuoso Layout Suite EXL and higher tiers 35

Supported Constraints and Parameters: Virtuoso Advanced Node for Layout Standard 38

Supported Constraints and Parameters: Virtuoso Advanced Node for Layout Only . 44

2

Antenna Constraints 51

antennaCreditRatio 52

antennaDiffusionWidthRatio 53

antennaModels 55

antennaModelCombo 61

antennaRatio 62

cumPerLayerAntennaRatio 67

diffWidthPlusFactor 72

Virtuoso Technology Data Constraints Reference

<u>maxFloatingArea</u>	75

3

Area Constraints

<u>maxCumAntenna</u>	82
<u>minArea</u>	84
<u>minAreaEdgeLength</u>	87
<u>minHoleArea</u>	92
<u>minIsolatedArea</u>	94
<u>minRectArea</u>	96

4

Density Constraints

<u>maxDensity</u>	102
<u>maxDensityCheck</u>	105
<u>maxDiffDensity</u>	108
<u>maxRelativeColorDensity</u>	111
<u>minDensity</u>	113
<u>minDensityCheck</u>	116
<u>minRelativeColorDensity</u>	119

5

Extension Constraints

<u>duyExmmntension</u>	125
<u>finExtension</u>	127
<u>lineEndAlignedWithCut</u>	129
<u>matchingFinIndex</u>	131
<u>maxExtension</u>	133
<u>minCenterlineExtension</u>	136
<u>minCornerExtension</u>	140
<u>minEndOfLineAdjExtension</u>	142
<u>minEndOfLineEdgeExtension</u>	144
<u>minEndOfLineExtension</u>	156
<u>minExtensionDistance</u>	159

Virtuoso Technology Data Constraints Reference

<u>minExtensionEdge</u>	165
<u>minExtensionEolSpacing</u>	184
<u>minExtensionOnLongSide</u>	186
<u>minExtensionToCenterLine</u>	189
<u>minExtensionToCorner</u>	194
<u>minGateExtension</u>	206
<u>minInnerVertexProximityExtension</u>	208
<u>minInsideCornerExtension</u>	212
<u>minNeighborExtension</u>	214
<u>minOppEndOfLineExtension</u>	227
<u>minOppExtension</u>	236
<u>minPRBoundaryExtension</u>	264
<u>minProtrusionWidthWithVia</u>	266
<u>minQuadrupleExtension</u>	269
<u>minSideExtension</u>	294
<u>minTouchingDirEnclosure</u>	299
<u>minViaExtension</u>	303
<u>minViaJointExtension</u>	305
<u>minVoltageExtension</u>	308
<u>minWireExtension</u>	312

6

<u>LEF Constraints</u>	315
<u>LEF_AREACAPACITANCE</u>	316
<u>LEF_CAPMULTIPLIER</u>	317
<u>LEF_EDGECAPACITANCE</u>	318
<u>LEF_HEIGHT</u>	319
<u>LEF_SHRINKAGE</u>	320

7

<u>Length and Width Constraints</u>	321
<u>allowedGateLengthRanges</u>	323
<u>allowedGateWidthRanges</u>	325
<u>allowedJogWidths</u>	327
<u>allowedLengthRanges</u>	332

Virtuoso Technology Data Constraints Reference

<u>allowedNeighborWidthRanges (One layer)</u>	337
<u>allowedNeighborWidthRanges (Two layers)</u>	341
<u>allowedNeighborWidthRangesOver</u>	344
<u>allowedPRBoundaryDimensions</u>	347
<u>allowedSpanLengthRanges</u>	349
<u>allowedWidthRanges</u>	354
<u>boundaryEOBlockage</u>	368
<u>illegalHGatePattern</u>	371
<u>maxDiagonalEdgeLength</u>	373
<u>maxDiffusionLength</u>	374
<u>maxEdgeLength</u>	376
<u>maxLength</u>	379
<u>maxLengthWithCut</u>	380
<u>maxNumMinEdges</u>	383
<u>maxPolyLength</u>	385
<u>maxTouchingDirectionLength</u>	387
<u>maxWidth</u>	389
<u>minChamferLength</u>	392
<u>minDiagonalEdgeLength</u>	397
<u>minDiagonalWidth</u>	399
<u>minEdgeAdjacentDistance</u>	400
<u>minEdgeAdjacentLength</u>	402
<u>minEndOfLineAdjacentToStep</u>	409
<u>minInsideCornerEdgeLength</u>	415
<u>minLength</u>	417
<u>minOutsideCornerEdgeLength</u>	419
<u>minPerimeter</u>	421
<u>minProtrusionWidthLength</u>	422
<u>minSize</u>	425
<u>minStepEdgeLength</u>	427
<u>minWidth</u>	437
<u>protrusionWidth</u>	441

8

<u>Miscellaneous Constraints</u>	443
<u>allowedShapeAngles</u>	445
<u>allowedSnapPatternDefs</u>	447
<u>allowedWireTypes</u>	448
<u>connectLayers</u>	449
<u>coreEOLBlockage</u>	450
<u>diagonalShapesAllowed</u>	452
<u>edgeMustCoincide</u>	453
<u>edgeMustOverlap</u>	455
<u>errorLayer</u>	459
<u>fixedMask</u>	460
<u>gateOrientation</u>	461
<u>ignoreShieldingOnLayers</u>	463
<u>implantGroup</u>	464
<u>integrationColorModel</u>	465
<u>layerDefaultColor</u>	466
<u>layerProp</u>	467
<u>layerThickness</u>	468
<u>lefLayerOrder</u>	469
<u>lefType</u>	470
<u>libProp</u>	471
<u>maxACCurrentDensity</u>	472
<u>maxNumCorners</u>	476
<u>minOneDArrayStructure</u>	477
<u>minStitchOverlap</u>	480
<u>msCrossTieLayers</u>	484
<u>msLayerCrossTieInterval</u>	485
<u>msSameMask</u>	486
<u>msShieldingLimit</u>	487
<u>msShieldStyle</u>	488
<u>multiMaskCheck</u>	490
<u>overlapViaGroup</u>	491
<u>pinConnectBlockage</u>	493
<u>preColoredLayers</u>	495

Virtuoso Technology Data Constraints Reference

<u>rectShapeDir</u>	496
<u>sameMaskOnLayer</u>	508
<u>trimMetalTrack</u>	509
<u>trimShape</u>	512
<u>useExistingShapesAsShield</u>	524
<u>validCutClass</u>	525
<u>vertexInsideForbidden</u>	527
<u>voltageMode</u>	529
<u>widthLengthRatio</u>	530

9

<u>NumCut Constraints</u>	533
<u>maxViaClusterNumCuts</u>	534
<u>minNumCut</u>	540
<u>minProtrusionNumCut</u>	549
<u>sameMetalAlignedCuts</u>	554

10

<u>Overlap Constraints</u>	565
<u>minDirectionalOverlap</u>	566
<u>minInsideCornerOverlap</u>	569
<u>minOverlapDistance</u>	571
<u>minWireOverlap</u>	573
<u>maxOverlap</u>	577
<u>overlapNotAllowed</u>	579

11

<u>Packaging Constraints</u>	581
<u>pkgMinAperture</u>	582
<u>pkgMinApertureDefault</u>	583
<u>pkgMinArea</u>	584
<u>pkgMinAreaDefault</u>	585
<u>pkgMaxBondwireLength</u>	586
<u>pkgMinBondwireLength</u>	587

Virtuoso Technology Data Constraints Reference

<u>pkgMinBondwireSpacing</u>	588
<u>pkgMinBondwireWidth</u>	589
<u>pkgMinHoleSpacing</u>	590
<u>pkgMaxLineWidth</u>	591
<u>pkgMinLineWidth</u>	592
<u>pkgMinSpacing</u>	593

12

Placement and Alignment Constraints 595

<u>allowedWidthSpacingPatternGroups</u>	596
<u>allowedWidthSpacingPatterns</u>	597
<u>defaultActiveWidthSpacingPattern</u>	598
<u>horizontalOffset</u>	599
<u>horizontalPitch</u>	600
<u>keepPRBoundarySharedEdges</u>	601
<u>keepSharedEdges</u>	603
<u>snapGridHorizontal</u>	605
<u>snapGridPeriod</u>	606
<u>snapGridVertical</u>	607
<u>verticalOffset</u>	608
<u>verticalPitch</u>	609

13

Routing Constraints 611

<u>clusterDistance</u>	613
<u>excludeLPPs</u>	614
<u>horizontalOffset</u>	615
<u>horizontalPitch (One layer)</u>	616
<u>ladderFrequency</u>	618
<u>ladderOffset</u>	620
<u>ladderStyle</u>	621
<u>layerMaskShiftAllowed</u>	623
<u>leftDiagOffset</u>	624
<u>leftDiagPitch</u>	626
<u>maxRoutingDistance</u>	628

Virtuoso Technology Data Constraints Reference

<u>numStrands</u>	629
<u>orthogonalSnappingLayer</u>	632
<u>orthogonalWSPGrid</u>	635
<u>rectangularGapMinSpacing</u>	637
<u>rightDiagOffset</u>	641
<u>rightDiagPitch</u>	642
<u>routingDirections</u>	643
<u>shieldGroupStyle</u>	645
<u>shieldMinLength</u>	646
<u>shieldTerminal</u>	647
<u>strandNumber</u>	648
<u>strandSpacing</u>	649
<u>strandWidth</u>	650
<u>styleAroundVia</u>	651
<u>tieShield</u>	652
<u>trackPattern</u>	653
<u>taperHalo</u>	654
<u>validLayers</u>	655
<u>validPurposes</u>	656
<u>validVias</u>	658
<u>verticalOffset</u>	661
<u>verticalPitch (One layer)</u>	662
<u>voltageLayerMarkerMapping</u>	664
<u>voltageLayerPairMarkerMapping</u>	665

14

<u>Spacing Constraints (One Layer)</u>	667
<u>allowedBetweenNeighborWidthRanges</u>	671
<u>allowedSpacingRanges (One layer)</u>	675
<u>bumpMinSpacing</u>	689
<u>cornerEndOfLineKeepout</u>	690
<u>cornerKeepOut</u>	693
<u>Values</u>	694
<u>cornerOnlyEndOfLineKeepout</u>	696
<u>cornerFillSpacing</u>	701

Virtuoso Technology Data Constraints Reference

<u>Group</u>	704
<u>cpodeGroupSpacing</u>	705
<u>deltaVLabelMapping</u>	706
<u>deltaVLayerMarkerMapping</u>	708
<u>deltaVLayerPairMarkerMapping</u>	710
<u>deltaVoltage</u>	713
<u>directionalEolKeepout</u>	714
<u>endOfLineKeepout</u>	717
<u>eolViaKeepout</u>	743
<u>forbiddenEdgePitchRange</u>	749
<u>forbiddenProximitySpacing</u>	766
<u>forbiddenSpacingRanges</u>	770
<u>forbiddenEdgePitchRange</u>	777
<u>forbiddenSpanLength</u>	783
<u>lineEndGap</u>	784
<u>lithoMacroHalo</u>	785
<u>maxFilling</u>	787
<u>maxTapSpacing</u>	789
<u>minAdjacentFourViaSpacing</u>	791
<u>minAdjacentTrimSpacing</u>	797
<u>minCenterToCenterSpacing</u>	799
<u>minClusterSpacing (One layer)</u>	801
<u>minConcaveCornerSpacing</u>	814
<u>minConvexCornerSpacing</u>	818
<u>minCornerSpacing (One layer)</u>	828
<u>minCornerToCornerDistance</u>	840
<u>minCornerVoltageSpacing</u>	844
<u>minDiagonalSpacing</u>	846
<u>minDiffNetSpacing</u>	849
<u>minEdgeLengthSpacing</u>	851
<u>minEndOfLineExtensionSpacing</u>	853
<u>minEndOfLinePerpSpacing</u>	860
<u>minEndOfLineSpacing</u>	864
<u>minEndOfLineToConcaveCornerSpacing</u>	902
<u>minEndOfLineToNotchSpacing</u>	905
<u>minEndOfNotchSpacing</u>	907

Virtuoso Technology Data Constraints Reference

<u>minExtensionSpacing</u>	910
<u>minFillToFillSpacing</u>	913
<u>minFillToShapeSpacing</u>	915
<u>minFiveWiresEndOfLineSpacing</u>	917
<u>minHoleWidth</u>	921
<u>minInfluenceSpacing</u>	923
<u>minJointCornerSpacing</u>	926
<u>minLargeNeighborViaArrayCutSpacing</u>	928
<u>minLengthWithCut</u>	931
<u>minNeighborVoltageSpacing (One layer)</u>	935
<u>minNotchSpacing</u>	938
<u>minNotchSpanSpacing</u>	948
<u>minOppSpanSpacing</u>	951
<u>minOrthogonalSpacing</u>	964
<u>minOuterVertexSpacing</u>	968
<u>minParallelSpanSpacing</u>	970
<u>minPRBoundaryExteriorHalo</u>	975
<u>minPRBoundaryInteriorHalo</u>	977
<u>minPrITwoSidesSpacing</u>	981
<u>minProtrusionSpacing</u>	984
<u>minSameNetSpacing (One layer)</u>	992
<u>minSideSpacing (One layer)</u>	994
<u>minSpacing (One layer)</u>	1011
<u>minSpanLengthExtensionSpacing</u>	1034
<u>minSpanLengthSpacing</u>	1036
<u>minStubInfluenceSpacing</u>	1060
<u>minVoltageSpacing (One layer)</u>	1063
<u>oneSideSpacing</u>	1066
<u>onGridOnly</u>	1068
<u>pgViaTrack</u>	1070
<u>requiredEndOfLineShape</u>	1072
<u>shapeRequiredBetweenSpacing</u>	1074
<u>sixWiresSpacing</u>	1076
<u>snapPatternDefOffset</u>	1078
<u>threeShapesForbiddenSpacingRanges</u>	1079
<u>trimMinAdjacentSpacing</u>	1083

<u>trimMinSpacing (One layer)</u>	1085
<u>viaKeepoutZone</u>	1095
<u>voltageLabelMapping</u>	1104
<u>voltageMarkerMapping</u>	1105

15

<u>Spacing Constraints (Two Layers)</u>	1107
<u>allowedSpacingRanges (Two layers)</u>	1109
<u>forbiddenExtensionRanges</u>	1113
<u>gateSpacingRanges</u>	1116
<u>maxSpacing</u>	1118
<u>minCenterLineSpacing (Two layers)</u>	1119
<u>minClusterSpacing (Two layers)</u>	1121
<u>minCornerSpacing (Two layers)</u>	1128
<u>minCutEdgeSpacing</u>	1130
<u>minCutRoutingSpacing (Two layers)</u>	1133
<u>minInnerVertexSpacing (Two layers)</u>	1146
<u>minInnerVertexSpacing (Three layers)</u>	1148
<u>minNeighboringShapesSpacing</u>	1150
<u>minNeighborVoltageSpacing (Two layers)</u>	1154
<u>minOppositeClearance (Two layers)</u>	1157
<u>minSameNetSpacing (Two layers)</u>	1159
<u>minSideSpacing (Two layers)</u>	1161
<u>minSpacing (Two layers)</u>	1170
<u>minSpacingOver</u>	1180
<u>minTouchingDirSpacing</u>	1183
<u>minVoltageSpacing (Two layers)</u>	1187
<u>shapeRequiredSpacing</u>	1191
<u>trimMinCutSpacing</u>	1195
<u>trimMinSpacing (Two layers)</u>	1198

16

<u>Via Construction Constraints</u>	1203
<u>allowedCutClass</u>	1205
<u>bothSidesViaForbiddenSpacingRange</u>	1212

Virtuoso Technology Data Constraints Reference

<u>checkOddViaLoop</u>	1220
<u>cutClassPreference</u>	1225
<u>definedCutClassesOnly</u>	1227
<u>forbiddenCutClassSpacingRange</u>	1228
<u>largeRectViaArrayAllowed</u>	1236
<u>maxViaArrayClusterSize</u>	1238
<u>maxViaStack</u>	1251
<u>mergedViaCornerToCornerSpacing</u>	1265
<u>minCutClassSpacing (One layer)</u>	1274
<u>minCutClassSpacing (Two layers)</u>	1295
<u>minDirectionalSpacing</u>	1308
<u>minEndOfLineCutSpacing</u>	1313
<u>minLargeViaArrayCutSpacing</u>	1319
<u>minLargeViaArraySpacing</u>	1324
<u>minLargeViaArrayWidth</u>	1329
<u>minNeighborViaSpacing</u>	1331
<u>minNestedViaSpacing</u>	1333
<u>minOrthogonalViaSpacing</u>	1336
<u>minParallelViaSpacing (One layer)</u>	1338
<u>minParallelViaSpacing (Two layers plus metal)</u>	1340
<u>minParallelWithinViaSpacing</u>	1342
<u>minSameMetalSharedEdgeViaSpacing</u>	1349
<u>minViaGroupSpacing</u>	1354
<u>minViaSpacing (One layer)</u>	1358
<u>minViaSpacing (Two layers)</u>	1389
<u>multiLayer</u>	1403
<u>redundantViaSetback</u>	1404
<u>stackable</u>	1406
<u>viaEdgeType</u>	1407
<u>viaGroup</u>	1415
<u>viaKeepoutMetal</u>	1419
<u>viaMeshEscapeLayer</u>	1422
<u>viaMeshMinCuts</u>	1423
<u>viaRequiredWithin</u>	1424
<u>viaSpacing</u>	1426
<u>viaStackingLimits</u>	1452

A

<u>Constraint Name Mapping for “minEnclosure”, “minExtension”, and “maxEnclosure”</u>	1457
<u>Implementation Specification</u>	1458

B

<u>Technology File Terms</u>	1461
<u>Connectivity Type</u>	1461
<u>Parallel Run Length</u>	1462
<u>Range</u>	1462
<u>Region-based Rule (One layer)</u>	1463
<u>Region-based Rule (Two layers)</u>	1464
<u>Routing Direction</u>	1465
<u>Spacing Direction</u>	1465
<u>Span Length</u>	1465
<u>Via</u>	1465
<u>Via Cut</u>	1465

C

<u>Load CDBA Type Tables</u>	1467
<u>2-D Legacy Table</u>	1468
<u>OA Style Table</u>	1470

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

This topic includes the following constraints in the following areas:

- Constraint Groups
- Constraints
- Constraint Parameters
- Advanced Nodes Constraints

Constraint Groups

Technology file syntax and the technology database together provide a standard mechanism for storing constraints or rules. However, all applications using a technology database may or may not support all the rules stored in the technology database.

Though there is a continuous effort to build rule support in applications as rules become available in the technology database, a time lag of an undetermined length typically occurs between the introduction of a new rule and the support for that new rule being added to an application.

There are three types of constraint groups that you can create:

- Foundry constraint group (foundry)
- Application-specific constraint group
- User-defined constraint group

There can only be one `foundry` constraint group, but there can be any number of application specific constraint groups and user-defined constraint groups for routing specific nets or net types.

The foundry, application-specific, and user-defined constraint groups may be created automatically during a translation process such as `lef2oa`, and is based on the information in LEF or DEF files. Other constraint groups may be created by your CAD group, and still others created by the individual layout designer. Some can be stored in a technology database and/or on the design itself.

In Virtuoso Layout Site, you typically have the following constraint groups as a required minimum.

- Foundry constraints group: The foundry constraints group has the absolute minimum rules which must be adhered to manufacture the design.

Setup constraints group: The `Setup` constraints group typically has the `virtuosoDefaultExtractorSetup` as the default value. Other possible values are `virtuosoDefaultSetup` and `LEFDefaultRouteSpec`.

To download a spreadsheet with constraint-to-application mapping information, click [Constraints_Map](#) (310 KB, ZIP file containing a Microsoft® Excel spreadsheet).

constraintGroups

```
constraintGroups (
  ( t_constraintGroupName [[b_override] [t_defName] [t_operator]]
    [memberConstraintGroups(t_memberConstraintGroupName ...)]
    [properties(l_properties)]
    t_constraintCategory(l_constraint)
  )
  ...
)
```

Enables you to define constraint groups. A constraint group is an ordered collection of constraints and constraint groups. Each constraint group can be assigned an operator that determines how constraints in that group are applied. A constraint group allows you to combine constraints to manage complex rules to be applied under different circumstances. This provides the flexibility to apply less or more stringent rules, as needed.

A constraint group can contain multiple specifications of the same constraint to set different values for that constraint. In such a scenario, by default, the first hard constraint in the constraint group takes precedence over any others. For more information about how constraints in a constraint group are applied, see [Combining Constraints in a Constraint Group](#).

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Parameters

t_constraintGroupName

The name of the constraint group.

Valid values: `foundry`, any unique string

b_override

Specifies whether the constraint group overrides other constraint groups.

Valid values: `t` (overrides), `nil` (does not override; default)

Note: This feature is not supported by Virtuoso. The parameter is a positional parameter that needs to be specified if you want to use any of the subsequent parameters.

t_defName

The name of the constraint group definition or type. For more information, see [Constraint Group Definitions](#).

t_operator

Specifies how constraints are evaluated when combined in a constraint group. All instances of a constraint defined on a specific layer (or layer pair) must be combined in one constraint group.

Valid values: `'precedence` (default), `'and`, `'or`

For more information about these operators, see [Combining Constraints in a Constraint Group](#).

t_memberConstraintGroupName

A list of constraint groups that are considered part of the constraint group.

Note: List member constraint groups in the order of precedence, starting with the highest precedence.

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

l_properties A list of properties associated with the constraint group. It has the following syntax:

```
( t_propName g_value ) ...
```

where,

t_propName is the name of the property.

g_value is the value assigned to the property.

Example:

```
;( group [override] )
;( ----- )
( "myConstraintGroup" nil
...
properties(
;( name value )
;( ---- )
( ndrsp "single" )
( ndrrp 6.700000 )
( ndrip 1 )
) ;properties
...
)
```

t_constraintCategory

The constraint category.

Valid values: spacings, orderedSpacings, spacingTables, routingGrids, placementGrids, antennaModels, viaStackingLimits, interconnect, allowedShapeAngles, routingDirections, colorControl

l_constraint

The constraint specification; as defined in [Constraints](#).

Constraint Group Definitions

The following values are supported for *t_defName*:

- foundry

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

The `foundry` constraint group specifies process rules typically provided by the foundry or manufacturer. It is stored in the technology database. Define in this constraint group the absolute minimum set of process rules required to manufacture a design.

The `foundry` constraint group is checked last in the constraint precedence lookup hierarchy. If a constraint override is not defined elsewhere, various applications use the values defined in the `foundry` constraint group.

All applications recognize the `foundry` constraint group. Constraints specified in the `foundry` constraint group must always be satisfied.

- ❑ **Creating the foundry constraint group:** The `foundry` constraint group is created and populated with information during `lef2oa` translation. The translators get the information from sections within the LEF file. Alternatively, you can create the foundry constraint group by loading an ASCII technology file containing the foundry constraint group information by using the *Tools – Technology File Manager* option in CIW.
- ❑ **Information in the foundry constraint group:** The `foundry` constraint group contains all the information necessary to manufacture the design. Everything from `minSpacing/minWidth` for layers, `minEnclosure`, `minStep`, to via rules, to electrical constraints such as antenna rules.
- ❑ **Using the `foundry` constraint group in Virtuoso:** The `foundry` constraint group is the final stop along the constraint precedence lookup trail. If a constraint override is not defined elsewhere, the tools use the value(s) in the foundry constraint group.

The `foundry` constraint group is usually not writable for most users, but can be viewed by using the Process Rule Editor or by taking an ASCII dump. You can launch the Process Rule Editor by clicking the RMB in the Constraint Manager assistant or by clicking the PRE icon from the toolbar in the Constraint Manager assistant. You can set the scope to Technology and view the foundry constraint group in this dialog box. See [Process Rule Editor](#).

■ `alternateFoundry`

(Virtuoso Advanced Node for Layout Only) The `alternateFoundry` constraint group contains one or more `foundry` constraint groups.

To enable the name of the constraint group that contains the foundry constraints, use the [AlternateFoundryCG](#) environment variable.

- `virtuosoDefaultExtractorSetup`: Specifies that it is used by default by the extractor for connectivity extraction. This constraint group can be stored within the technology database or directly on the design itself, that is, `lib/cell/view`.

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

- ❑ Information in the `virtuosoDefaultExtractorSetup` Constraint Group: The `virtuosoDefaultExtractorSetup` constraint group includes layers needed for extracting connectivity, plus poly, all metal layers, and cut layers in the valid layers section.
- ❑ Using the `virtuosoDefaultExtractorSetup` Constraint Group in Virtuoso: A constraint group for extraction is required to specify the extractable layers.

Connectivity extraction gets the constraint group name from the value of the `setupConstraintGroup` variable in your `.cdsenv` file. The default value is `virtuosoDefaultExtractorSetup`. You can change the name of the constraint group by resetting the value. If the `setupConstraintGroup` variable has no value set, the extractor prompts you for a constraint group to use for extraction.

- `virtuosoDefaultSetup`: Supplies a default set of constraints for both interactive and automatic routing.
 - ❑ Information in the `virtuosoDefaultSetup` Constraint Group: The `virtuosoDefaultSetup` constraint group contains `validVias` and `validLayers` for routing (no cut layers), and possibly other rules such as `minWidth` and `minSpacing` for the layers, overriding the foundry rules.
 - ❑ Using the `virtuosoDefaultSetup` Constraint Group in Virtuoso: The `virtuosoDefaultSetup` constraint group is typically used to define a full set of `validLayers` and `validVias` for the purpose of default interactive and automatic routing.

For example, if any particular signal is selected for routing and does not have a constraint group assigned to it, the constraints in the `virtuosoDefaultSetup` constraint group apply. The contents of `validLayers` in `virtuosoDefaultSetup` are usually a subset of those defined in the `virtuosoDefaultExtractorSetup` constraint group because, while you may want to extract connectivity for non-metal layers, you do not want interactive and automatic routing to use the non-metal layers.

- `virtuosoExtractorSetup`
- `virtuosoMPTSetup`
- `LEFDefaultRouteSpec`: Stores routing information, such as information about layers and vias. This constraint group contains rule definitions typically used in digital standard cell applications or custom and mixed-signal applications that require metal layers only.
 - ❑ Creating the `LEFDefaultRouteSpec` Constraint Group: The `LEFDefaultRouteSpec` constraint group is specific to a LEF or DEF design flow and is created during `lef2oa` translation. Since a LEF file serves as the library source for technology information, the `lef2oa` translator puts all the technology information into the `LEFDefaultRouteSpec` constraint group. The valid layers,

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

valid via definitions, and valid rules also become part of the `LEFDefaultRouteSpec` constraint group.

The `lef2oa` translator translates any `NONDEFAULTRULES` in LEF as constraint groups in the OpenAccess tech database.

- ❑ **Information in the `LEFDefaultRouteSpec` Constraint Group:** The `LEFDefaultRouteSpec` contains valid layer and valid via definitions. The via definitions are typically standard vias and several custom vias used by other routers such as NanoRoute in the Innovus environment. The `LEFDefaultRouteSpec` also contains any other non-default rules such as wide metal rules.
- ❑ **Using the `LEFDefaultRouteSpec` Constraint Group in Virtuoso:** You can select the `LEFDefaultRouteSpec` as the default constraint group for interactive or automatic routing in the Wire Editing section of the Layout Editor Options form (*Options — Editor*).

- `LEFSpecialRouteSpec`: Stores information about power routing vias.
- `taper`
- `inputTaper`
- `outputTaper`
- `shielding`

Application-specific constraint groups, including all of the above barring `foundry` and `alternateFoundry`, influence the behavior of a particular application. You can also create application-specific constraint groups with user-defined names. To designate such constraint groups as application-specific, assign to them an appropriate constraint group definition, which determines the type of objects to which the constraint group applies.

User-Defined Constraint Groups

User-defined constraint groups help combine constraints by using `'and` or `'or` operators. They are referenced from the `foundry` constraint group by using the `memberConstraintGroups` keyword.

You can define only one `foundry` constraint group in a technology database, but there is no such restriction on the number of application-specific and user-defined constraint groups that you can create. However, all constraint groups defined in a given technology database must be uniquely named.

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Example

```
constraintGroups (
; ( group      [override]      [definition]      [operator] )
; ( ----- )
( "virtuosoDefaultExtractorSetup" nil "virtuosoExtractorSetup"
  interconnect (
    ( (validVias (VIA12 VIA23 VIA34) )
    ) ;interconnect
  ) ;virtuosoDefaultExtractorSetup
) ;constraintGroups
```

Note: For a constraint group with name `LEFDefaultRouteSpec`, the technology file loader automatically uses the constraint group definition `LEFDefaultRouteSpec`, and for a constraint group with name `LEFSpecialRouteSpec`, the technology file loader automatically uses the constraint group definition `LEFSpecialRouteSpec`.

Combining Constraints in a Constraint Group

Constraint groups use an operator to determine how the constraints in the group are applied. There are three constraint group operators. By default, constraints in a constraint group are applied based on precedence.

- `'precedence` (default): Specifies that constraints or member constraint groups are evaluated based on the order in which they are listed in a constraint group. For each constraint type, the first hard constraint found in a constraint group must be met. If the first hard constraint cannot be met, no other constraint of that type in the constraint group is considered.

If a soft constraint of a given type is found, it is enforced, if possible. Otherwise, the next soft constraint in the group is evaluated.

Only one constraint of a particular type in the group applies, and the first hard or soft constraint that is met satisfies the constraint group evaluation.

Note: Constraint groups that use the `'precedence` operator can have other constraint groups as members.

- `'and`: Specifies that all constraints in the constraint group must be met. All applicable constraints are checked. If a violation is found, the remaining applicable constraints are not checked.

A constraint is applicable for a given set of geometry if the parameters associated with the constraint are met. For example, the constraint below is applicable only between two shapes if at least one of those shapes has an area of less than 0.03 microns.

```
spacings (
  ( minSpacing "Metal2"
    'area 0.03
    0.2
```

```
)  
) ;spacings
```

Note: Constraint groups that use the 'and operator cannot have any member constraint groups.

- 'or: Specifies that only one constraint in the constraint group must be satisfied, that is, of all applicable constraints, at least one should be met. A violation is reported if all applicable constraints are not met.

Limited support is available for the 'or operator in Design Rule Driven (DRD) editing, where it is used mainly to combine constraints of type extension.

Note: Constraint groups that use the 'or operator cannot have any member constraint groups.

Determining Constraint Group Precedence

A technology database can contain multiple constraint groups that define the same constraints. An application searches the constraint groups it recognizes in the order of precedence it has set and applies the first hard constraint it finds. If the application searches all of the constraint groups without finding a hard constraint, it applies the first soft constraint that it finds.

Constraint groups are applied in the following order of precedence:

1. The constraint group being referenced by a command that is currently active in the design window.
2. A constraint group recognized as the constraint group of first precedence by an application. Refer to your application documentation to determine which constraint groups the application recognizes and the precedence it applies to them.
3. The `foundry` constraint group. All applications recognize the `foundry` constraint group.

Managing Member Constraint Groups

You can establish a constraint group, and then specify it as a member of another constraint group. When multiple member constraint groups are specified, their order of precedence is from left to right, that is, the first has the highest precedence and the last the lowest.

Example

```
constraintGroups  
  ( "protrusionGroup" nil nil 'and
```

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

```
    spacings (
      ( minProtrusionNumCut "cut1"
        'distance 0.50 'length 2.0 'width 1.0
        'distanceWithin 1.2 2
      )
      ( minProtrusionNumCut "cut1"
        'distance 0.75 'length 3.0 'width 1.5
        'distanceWithin 1.2 3
      )
    ) ;spacings
  ) ;protrusionGroup
  ( "foundry" nil
    memberConstraintGroups("protrusionGroup")
  ) ;foundry
) ;constraintGroups
```

Constraints

A constraint can be specified in any constraint group and must be:

- Specified according to the syntax delineated in this reference manual
- Used in accordance with the constraint definition provided in this reference manual

Note: User-defined constraints are maintained in the technology file, but are not used or recognized by Cadence applications. These constraints are maintained in the technology file to preserve existing Pcell code that might refer to them. Most importantly, a constraint must have the proper name and syntax for it to be recognized by applications and commands that are designed to use the constraint and to be interoperable with other tools that run on OpenAccess.

All constraints have the same general syntax. Only the number of layers and parameters varies for each constraint.

```
( t_constraintName ["t_layerName" | x_layerNumber | ("t_layerName"
    "t_purposeName")] ...]
    [parameter ...]
    g_value
    ['soft | 'hard]
    ['ref t_reference]
    ['description t_description]
)
```

Parameters

t_constraintName The name of the constraint.

"t_layerName" | x_layerNumber | ("t_layerName" "t_purposeName")

A list of 0 to 3 layers, depending on the constraint.

You can specify either the layer name or number or the layer-purpose pair name. If specifying the layer-purpose pair name, both layer and purpose names must be enclosed individually in double quotes and separated by a space. Additionally, the layer-purpose pair must be enclosed in parentheses.

Type: String (layer and purpose names) or Integer (layer number)

parameter A list of 0 or more parameters, depending on the constraint.

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

<code>g_value</code>	The constraint value. Either a scalar value, a list, or a table, depending on the constraint.
<code>'soft 'hard</code>	Specifies whether a constraint must be absolutely met, or whether it is only a recommended value. For more information, see Constraint Parameters .
<code>'ref</code>	Lets you add a reference ID at the end of a constraint. The reference ID can be used to identify that constraint specification. This parameter can be specified for any constraint. It is used in violation markers and in the Annotation Browser.
<code>'description</code>	Lets you add a description at the end of a constraint. This parameter can be specified for any constraint. The description is displayed in violation markers and in the Annotation Browser.

Example

```
orderedSpacings(  
  ( minOppExtension "Metall" "Vial"  
    'cutDistance 0.1      ;first parameter  
    'noSharedEdge        ;second parameter  
    (2.1 0.7)            ;constraint value  
    'soft  
    'ref "M1.EX.1"  
    'description "Metall extension past Vial"  
  )  
) ;orderedSpacings
```

Spacing Table Constraints

Spacing tables allow conditional evaluation and application of a constraint value by means of lookup tables. The spacing tables can apply to one or two layers or can be one- or two-dimensional, with the constraint value determined based on one or two other design values. The table specification in a constraint definition takes the following syntax:

For a one-dimensional table: (*g_index g_value ...*)

For a two-dimensional table: (*g_index1 g_index2 g_value ...*)

where, an index is a design parameter value used to determine the constraint value.

Table entries must be specified in the ascending order of index value. If they are not specified in ascending order, the compiler sorts them in ascending order, as is reflected in a dump of the technology database.

For a constraint value to apply, the design value must be "greater than or equal to" the index value. In the following example, if the width is greater than or equal to 0, the constraint value is 10, and if the width is greater than or equal to 3, the constraint value is 15.

```
spacingTables(
  ( minSpacing "Metall1"
    (( "width" nil nil ))
    (
      (0   10)
      (3   15)
    )
  )
) ;spacingTables
```

The first value in the table that matches a query during lookup is returned.

For two-dimensional tables, all table entries must be explicitly specified; otherwise, the technology file compiler uses the default value for the unspecified entries. If no default value is specified, the compiler adds the 0 for the missing table entries.

	b1	b2	b3
a1	v11	v12	<i>default</i>
a2	v21	<i>default</i>	v23
a3	v31	v32	<i>default</i>

Ensure that all tables (both one- and two-dimensional) only have entries of interest (that is, the index that affects the rule). Adding extra index values, particularly if they are large, can adversely affect the performance while checking.

Virtuoso Technology Data Constraints Reference

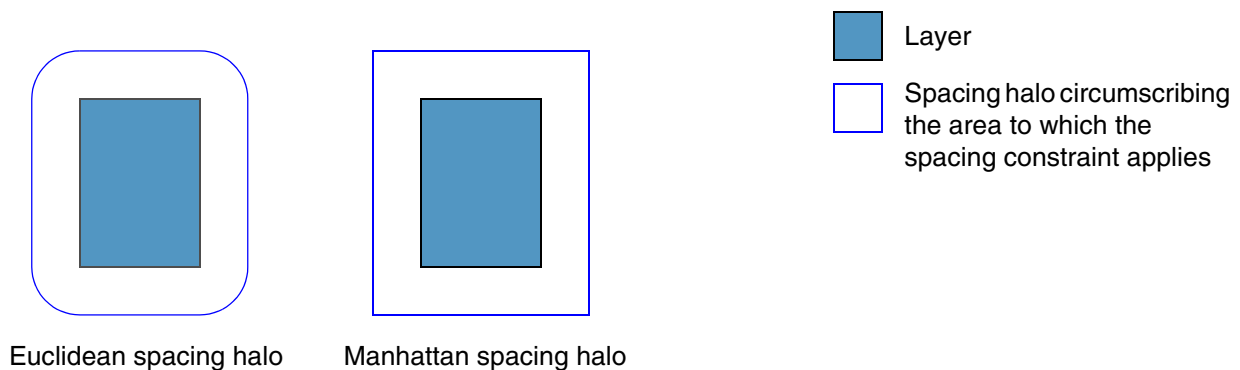
Constraint Groups and Constraints

Constraint tables do not support expressions for indexes; you cannot parameterize an index by specifying it as a `techParams` value (`techParam(paramName)`).

Euclidean and Manhattan Spacing Constraints

By default, spacing constraints apply within the Euclidean distance around a shape, which means that 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 reference manual, constraints for which you can specify Manhattan spacing have `'manhattan'` specified in their syntax.



Constraint Parameters

Constraint parameters can be placed on a constraint at the end of the constraint specification.

'hard

```
'hard
```

Identifies the constraint as a hard constraint that must be satisfied. This parameter can be specified for any constraint.

Example

```
orderedSpacings(  
    ( minOppExtension "poly1" "metal2" (2.1 0.7) 'hard )  
);orderedSpacings
```

'soft

```
'soft
```

Identifies the constraint as a soft or preferred constraint. A soft constraint establishes the preferred constraint definition, but is not absolutely enforced like a hard constraint. This parameter can be specified for any constraint.

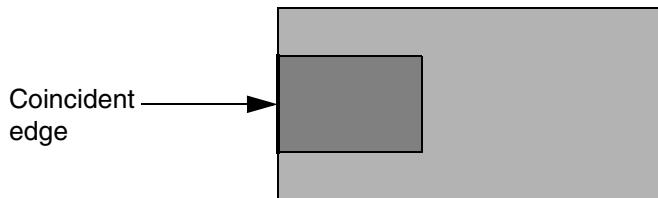
Example

```
orderedSpacings(  
    ( minOppExtension "poly1" "metal2" (2.1 0.7) 'soft )  
);orderedSpacings
```


'coincidentAllowed

'coincidentAllowed

Allows abutted devices with coincident edges. The constraints to which this parameter applies specify it as an option in the syntax.



Example

```
orderedSpacings(  
    ( minOppExtension "poly1" "metal2" (2.1 0.7) 'coincidentAllowed )  
) ;orderedSpacings
```

'description

'description *t_desc*

Adds a description for a constraint. This parameter can be specified for any constraint.

Arguments

<i>t_desc</i>	Description to be associated with the specified constraint for use by applications. Valid values: Any string

Note: Some applications display this description with an error message.

Example

```
orderedSpacings(  
    ( minOppExtension "poly1" "metal2" (2.1 0.7)  
      'description "This is a constraint description."  
    )  
) ;orderedSpacings
```

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

'ref

`'ref t_ref`

Adds a reference ID to a constraint. The reference ID can be used to identify a constraint specification. This parameter can be specified for any constraint.

Arguments

<code>t_ref</code>	Reference ID to be associated with the constraint specification. Valid values: Any string

Example

```
orderedSpacings(  
  ( minOppExtension "poly1" "metal2" (2.1 0.7)  
    'ref "const1S"  
  )  
) ;orderedSpacings
```

'manhattan

`'manhattan`

Specifies that the constraint adheres to Manhattan spacing, instead of the default Euclidean spacing.

Example

```
spacings(  
  ( minSpacing "poly1" 2.1 'manhattan )  
) ;spacings
```

Advanced Nodes Constraints

There are two levels of advanced nodes constraints depending on the licenses they use:

- Virtuoso Layout Suite EXL and higher tiers

To use these constraints, you need the Virtuoso Advanced Node Option for Layout (95511) and Virtuoso Layout Suite EXL (95800) licenses.

- Virtuoso Advanced Node for Layout Standard

To use these constraints, you must have either the Virtuoso_Adv_Node_Opt_Lay_Std (95512) or Virtuoso_Adv_Node_Opt_Layout (95511) license.

- Virtuoso Advanced Node for Layout Only

To use these constraints, you must have the Virtuoso_Adv_Node_Opt_Layout (95511) license. The Virtuoso_Adv_Node_Opt_Lay_Std (95512) cannot be used.

Note: For constraints listed with parameter names, the specified licensing is required only if those parameters are set. If only the constraint name is listed, any setting of the constraint requires the specified license.

For more information about advanced node licensing, see *License Requirements for Advanced Node Features* in *Virtuoso Software Licensing and Configuration User Guide*.

Supported Constraints and Parameters: Virtuoso Layout Suite EXL and higher tiers

Constraint Name	Parameters
<u>allowedWidthRanges</u>	exceptOverlap
<u>antennaCreditRatio</u>	
<u>antennaDiffusionWidthRatio</u>	
<u>boundaryEOLBlockage</u>	p1, within, distance
<u>coreEOLBlockage</u>	
<u>diffWidthPlusFactor</u>	
<u>directionalEolKeepout</u>	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>endOfLineKeepout</u>	exceptSameSideMetal, exceptNoOppSideMetal, exceptSpacing, withinCut, cutLayers, exceptEolSpacing, horizontal, vertical, exceptLineEndGap, eolSpacingRange, otherEolSpacingRange, jointAsEol, checkOtherWidth, equalRectWidth
<u>eolViaKeepout</u>	
<u>forbiddenSpacingRanges</u>	ignoreMiddle
<u>implantGroup</u>	
<u>lineEndGap</u>	
<u>maxCumAntenna</u>	lowerLayer, upperLayer
<u>maxLength</u>	width
<u>mergedViaCornerToCornerSpacing</u>	tripletCenterViaLarger, tripletCenterViaSpacingX, tripletCenterViaSpacingY, lengthRange, otherLengthRange
<u>minAdjacentFourViaSpacing</u>	exceptDiffMetal, mergedCuts, minCorner, minInnerCorner, maxInnerCorner, exceptOverlap, exceptMinLength, cutWithin, exceptOtherWidthWithCut, otherLayer, width
<u>minClusterSpacing (One layer)</u>	enclosingLayer, viaEdgeType
<u>minConvexCornerSpacing</u>	lowCornerSpacing, middleWires
<u>minCornerToCornerDistance</u>	extensionLength, oppositeDirection
<u>minCornerSpacing (One layer)</u>	cornerToCorner
<u>minCornerVoltageSpacing</u>	endOfLineWidth
<u>minCutRoutingSpacing (Two layers)</u>	horizontal vertical, toDiffConcaveCorner

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minDirectionalSpacing</u>	excludeCutEdge, minNumCuts
<u>minEndOfLineEdgeExtension</u>	exceptSpacing, minVoltage, maxVoltage
<u>minLengthWithCut</u>	
<u>minOppExtension</u>	otherExtension, widthHorizontal widthVertical
<u>minParallelWithinViaSpacing</u>	otherWidth
<u>minPRBoundaryInteriorHalo</u>	exceptLayer, exceptWidth
<u>minQuadrupleExtension</u>	otherMask1, otherMask2, otherMask3, exactZero, otherWithinWidthTable, otherWithinWidthHorizontal otherWithinWidthVertical, otherSideTable, lineEndGapOnlyTable
<u>minSideSpacing (One layer)</u>	mask1, mask2, mask3, otherLayer, cutClass, cutWithin
<u>minSpacing (One layer)</u>	length, intersectLayers
<u>minSpanLengthSpacing</u>	diffMask
<u>minSpacing (Two layers)</u>	length
<u>minStepEdgeLength</u>	jogEdge, adjJogEdge, adjNotchEdge, adjEolSpacingRange
<u>minViaGroupSpacing</u>	mergedCuts, otherLayer
<u>minViaSpacing (One layer)</u>	deltaVoltage, exceptGap, exceptSameMetalOverlap, orthogonalEnclosingMetal, otherCutClass, otherViaEdgeType, fixedCutClassList, otherFixedCutClassList, horizontalSpacingRange, verticalSpacingRange, farEnds

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minViaSpacing (Two layers)</u>	deltaVoltage
<u>minWidth</u>	length
<u>overlapViaGroup</u>	
<u>pinConnectBlockage</u>	
<u>rectShapeDir</u>	exceptWidth, widthHorizontal widthVertical
<u>requiredEndOfLineShape</u>	
<u>sameMetalAlignedCuts</u>	cutClassSetList, noViaMetalOverlap, otherLayer
<u>trimMetalTrack</u>	halfInFiller
trimMinCutSpacing	
<u>viaEdgeType</u>	metalWithin, noMetalWithin, horizontalEdge verticalEdge, exceptSpacing, sameSide, noMetalLowWithin
<u>viaGroup</u>	allowBend, disallowHorizontal disallowVertical
<u>viaKeepoutMetal</u>	
<u>viaKeepoutZone</u>	viasAnyConnectivity, viasOnSameMetal, viasOnDiffMetal
<u>viaSpacing</u>	width, exceptLineEndGap

Supported Constraints and Parameters: Virtuoso Advanced Node for Layout Standard

Constraint Name	Parameters
<u>allowedCutClass</u>	
<u>allowedLengthRanges</u>	
<u>allowedNeighborWidthRanges (One layer)</u>	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>allowedNeighborWidthRanges</u> <u>(Two layers)</u>	
<u>allowedNeighborWidthRangesOver</u>	
<u>allowedPRBoundaryDimensions</u>	
<u>allowedSpacingRanges</u> (One layer)	widthRanges, exceptOverLayer, overLayerPrl, overLayerWidthRanges, exceptNumShapes, numShapeDistance, numShapeWidthRanges, numShapeMaxWidth, stepSize, stepRange, overLayer
<u>allowedWidthRanges</u>	measureHorizontal, measureVertical, stepRange, stepSize
<u>edgeMustCoincide</u>	
<u>edgeMustOverlap</u>	
<u>endOfLineKeepout</u>	orGroupID, exceptSideWithin, exceptFromFrontEdge, exceptFromBackEdge
<u>forbiddenCutClassSpacingRange</u>	
<u>forbiddenEdgePitchRange</u>	
<u>forbiddenProximitySpacing</u>	
<u>layerDefaultColor</u>	
<u>lithoMacroHalo</u>	
<u>maxCumAntenna</u>	
<u>maxNumCorners</u>	
<u>maxViaStack</u>	minMultiCut
<u>minCenterLineSpacing</u> (Two layers)	
<u>minClusterSpacing</u> (One layer)	
<u>minClusterSpacing</u> (Two layers)	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minCornerSpacing (One layer)</u>	
<u>minCornerSpacing (Two layers)</u>	
<u>minCutClassSpacing (One layer)</u>	bothNegativePrIs, nonCutClassEdgeSpacing, sameMask
<u>minCutClassSpacing (Two layers)</u>	nonZeroEnclosure, nonCutClassEdgeSpacing, cutClassSizeBy
<u>minCutRoutingSpacing (Two layers)</u>	enclosingLayerWidth, enclosingLayer, maskOverlap, parallelExtension, prl, parallelEdgeWithin, otherExtension, anyOppositeExtension, viaEdgeType
<u>minDirectionalOverlap</u>	
<u>minEdgeAdjacentLength</u>	concaveCorner
<u>minEndOfLineAdjacentToStep</u>	adjacentLength, concaveCorner
<u>minEndOfLineExtensionSpacing</u>	negativePRL, sameMask
<u>minEndOfLineSpacing</u>	exactEolWidth, horizontal, vertical, extendBy, sizeBy, widthRanges, sameMask, diffMask, sameNet, sameMetal, exceptExactAligned, bothWires, wrongDirSpace
<u>minEndOfLineToConcaveCornerSpacing</u>	dualAdjacentLength
<u>minEndOfLineToNotchSpacing</u>	
<u>minExtensionDistance</u>	horizontal, vertical, cutClass, exceptEdgeLengthRange, insidePurposes, outsideLayers, outsidePurposes

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minExtensionEdge</u>	exceptConcaveCorner, twoSides, horizontal, vertical, insidePurposes, outsideLayers, outsidePurposes
<u>minExtensionOnLongSide</u>	
<u>minExtensionToCenterLine</u>	
<u>minExtensionToCorner</u>	
<u>minInnerVertexProximityExtension</u>	
<u>minJointCornerSpacing</u>	sameMask
<u>minMeshEscapeLayer</u>	
<u>minMeshMinCuts</u>	
<u>minNeighborExtension</u>	
<u>minNeighboringShapesSpacing</u>	
<u>minNotchSpacing</u>	notchWidth
<u>minNumCut</u>	bothAboveBelow, sameMetalOverlap
<u>minOppExtension</u>	horizontal, vertical, includeAbutted, stepSizePair, exactPrl, hasExactPrl, insideLayers, insidePurposes, outsideLayers, outsidePurposes, alignedLayer, alignedHorizontal, alignedVertical
<u>minParallelWithinViaSpacing</u>	cutClass, longEdgeOnly
<u>minPRBoundaryInteriorHalo</u>	horizontal, stepSize, vertical
<u>minProtrusionSpacing</u>	excludeSpacing
<u>minProtrusionWidthLength</u>	enclosedWidth
<u>minProtrusionWidthWithVia</u>	
<u>minRectArea</u>	maxWidth

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minSideExtension</u>	exceptEdgeLengthRanges
<u>minSideSpacing (One layer)</u>	
<u>minSideSpacing (Two layers)</u>	
<u>minSpacing (One layer)</u>	exceptEolWidth, ignoreShapesInRanges, sameMask
<u>minSpanLengthSpacing</u>	
<u>minStepEdgeLength</u>	any, horizontalEdge, verticalEdge, allCorner, concaveCorner, convexCorner, mixedCorner, exceptExactLength, exceptExactBothLength
<u>minStitchOverlap</u>	
<u>minViaSpacing (One layer)</u>	horizontal, vertical, bothCuts, exceptExactAligned, enclosingLayer, overLayer, overLayerWidth, viaEdgeType, sameMask, sizeBy, diffMask, orthogonalEdgePair, facingEdges, exceptSameNet, exceptSameMetal
<u>minWireOverlap</u>	
<u>onGridOnly</u>	
<u>pgViaTrack</u>	
<u>rectShapeDir</u>	extendBy, exceptViaLayer, exceptViaSize, exceptExactSize, exceptEdgeLength, twoSides, width, widthRanges, exceptNonCorePins, insideLayers, insidePurposes, outsideLayers, outsidePurposes
<u>shapeRequiredBetweenSpacing</u>	
<u>viaRequiredWithin</u>	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>viaSpacing</u>	allCuts, cutSizeRanges, exceptOppositeCornerNeighbors, twoCuts
<u>voltageLayerMarkerMapping</u>	
<u>voltageLayerPairMarkerMapping</u>	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Supported Constraints and Parameters: Virtuoso Advanced Node for Layout Only

Constraint Name	Parameters
<u>allowedBetweenNeighborWidthRanges</u>	
<u>allowedCutClass</u>	minVoltage, maxVoltage
<u>allowedSnapPatternDefs</u>	
<u>allowedSpanLengthRanges</u>	otherSpanLength, otherSpanTable
<u>allowedWireTypes</u>	
<u>bothSidesViaForbiddenSpacingRange</u>	
<u>antennaModelCombo</u>	
<u>boundaryEOLBlockage</u>	
<u>cornerEndOfLineKeepout</u>	
<u>cornerOnlyEndOfLineKeepout</u>	
<u>cutClassPreference</u>	
<u>endOfLineKeepout</u>	mask1, mask2, mask3, diffMask, forwardGap, otherEndEol, twoSides, minEolWidth
<u>forbiddenCutClassSpacingRange</u>	longEdge, sameMask, allCuts, layer, layerWidth, mask1, mask2, mask3, paraLength, within
<u>forbiddenEdgePitchRange</u>	minWidth, outerSameMask, outerWithin, outerWidth, mask1, mask2, mask3
<u>forbiddenSpacingRanges</u>	
<u>layerMaskShiftAllowed</u>	
<u>maxDensityCheck</u>	
<u>maxEdgeLength</u>	
<u>maxRelativeColorDensity</u>	
<u>maxViaArrayClusterSize</u>	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>maxViaStack</u>	exceptPGNet, noViaList, noCutClassList, withinLayerName, withinList, exceptCutClassList
<u>mergedViaCornerToCornerSpacing</u>	
<u>minAdjacentFourViaSpacing</u>	
<u>minClusterSpacing (One layer)</u>	clusterToCluster, edgeToEdge, sameMask, widthHorizontal widthVertical
<u>minConcaveCornerSpacing</u>	
<u>minCornerSpacing (One layer)</u>	useEdgeLength, sameMask, exceptNotchLength
<u>minCornerVoltageSpacing</u>	
<u>minCutClassSpacing (One layer)</u>	horizontalOverlap, verticalOverlap
<u>minCutClassSpacing (Two layers)</u>	minEnclosure
<u>minCutEdgeSpacing (Two layers)</u>	
<u>minCutRoutingSpacing (Two layers)</u>	shortEdgePrl, deltaVoltage
<u>minDensityCheck</u>	
<u>minDirectionalSpacing</u>	
<u>minEdgeAdjacentLength</u>	threeConcaveCorners, width
<u>minEndOfLineEdgeExtension</u>	minWidth, shortEdgeOnly
<u>minEndOfLineExtensionSpacing</u>	nonEol, otherWidth
<u>minEndOfLinePerpSpacing</u>	negativePRL
<u>minEndOfLineSpacing</u>	exceptExactEolWidth, fillConcaveCorner, negativePRL, parallelSameMask
<u>minExtensionEdge</u>	edgeExtension, layer
<u>minFiveWiresEndOfLineSpacing</u>	
<u>minNeighborExtension</u>	mask1, mask2, mask3, mask4

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minNestedViaSpacing</u>	
<u>minNotchSpacing</u>	excludeSpacing, horizontal, vertical
<u>minNumCut</u>	fullyEnclosed
<u>minOppositeClearance (Two layers)</u>	
<u>minOppEndOfLineExtension</u>	horizontal, vertical
<u>minOppExtension</u>	extraOnly, jogLengthOnly, horizontalJog, verticalJog, jogWireWidth, treatLAsJog, hollowHorizontal, hollowVertical
<u>minOrthogonalSpacing</u>	
<u>minPrlTwoSidesSpacing</u>	
<u>minQuadrupleExtension</u>	mask1, mask2, mask3, otherLayer, useMaxWidth, useMinWidth, withinFirstWidth, directionalExtension, maxLength, parallelWithinTable, trimLayer, trimLengthTable, includeAbuttedTable
<u>minRectArea</u>	mask1, mask2, mask3, layer, overlapType, insideLayers, insidePurposes, outsideLayers, outsidePurposes
<u>minRelativeColorDensity</u>	
<u>minSideSpacing (One layer)</u>	cornerEuclidian, lengthRanges, otherLengthRanges, prlRange, sameMask, mask1, mask2, mask3, widthHorizontal widthVertical, otherFacingEdgeRange, facingEdgeHorizontal, facingEdgeVertical

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minSideSpacing (Two layers)</u>	cornerEuclidian, exceptOverlap, mask1, mask2, mask3, otherMask1, otherMask2, otherMask3, deltaVoltage, prlRange
<u>minSpacing (One layer)</u>	fillConcaveCorner, mask1, mask2, mask3, ignoreIntermediateShapes, cornerAllowed
<u>minSpacing (Two layers)</u>	crossingAllowed, cornerAllowed
<u>minSpacingOver (Two layers)</u>	mask1, mask2, mask3
<u>minSpanLengthExtensionSpacing</u>	
<u>minSpanLengthSpacing</u>	exactSelfSpacingTable, eolExceptionNoPrl, minSpanForExceptEolWidth, minSpanSpacingRangesPrl, sameMask, spacingToMinSpanTable, useEdgeLength
<u>minStepEdgeLength</u>	extendToAlign
<u>minViaGroupSpacing</u>	groupToGroup

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>minViaSpacing (One layer)</u>	above, below, bothAboveBelow, diffMask, sizeByTouchedCorners, includedEdgeAligned, alignedHorizontal alignedVertical, shortSideToShortSide shortSideToLongSide shortSideToAnySide longSideToLongSide longSideToAnySide, prlRange, viasOnSameNet viasOnSameMetal viasOnSameVia, exceptViasOnSameNet exceptViasOnSameMetal exceptViasOnSameVia
<u>minViaSpacing (Two layers)</u>	exceptSameNet, exceptSameMetal, horizontal vertical, oppositeSides, prl, prlRange
<u>minVoltageExtension</u>	
<u>minVoltageSpacing (One layer)</u>	horizontal, vertical
<u>minVoltageSpacing (Two layers)</u>	horizontal, vertical
<u>multiMaskCheck</u>	
<u>multiLayer</u>	
<u>oneSideSpacing</u>	
<u>onGridOnly</u>	exceptWidth
<u>orthogonalSnappingLayer</u>	
<u>orthogonalWSPGrid</u>	
<u>preColoredLayers</u>	
<u>rectangularGapMinSpacing</u>	
<u>sameMaskOnLayer</u>	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Constraint Name	Parameters
<u>sameMetalAlignedCuts</u>	withCut, centerToCenterSpacing, edgeToEdgeSpacing, horizontal, vertical, sameViaConnectivity
<u>snapPatternDefOffset</u>	
<u>trimMetalTrack</u>	
<u>trimMinAdjacentSpacing</u>	
<u>trimMinSpacing (One layer)</u>	
<u>trimMinSpacing (Two layers)</u>	
<u>trimShape</u>	
<u>vertexInsideForbidden</u>	
<u>viaGroup</u>	
<u>viaKeepoutZone</u>	
<u>viaSpacing</u>	minDistance, sameMask, sameNeighborCuts, exceptWithin, edgeExtension, layer, horizontal, vertical, twoCutsSpacing
<u>widthLengthRatio</u>	

Virtuoso Technology Data Constraints Reference

Constraint Groups and Constraints

Antenna Constraints

This topic includes the following constraints:

- [antennaCreditRatio](#)
- [antennaDiffusionWidthRatio](#)
- [antennaModels](#)
- [antennaModelCombo](#)
- [antennaRatio](#)
- [cumPerLayerAntennaRatio](#)
- [diffWidthPlusFactor](#)
- [maxFloatingArea](#)

antennaCreditRatio

```
spacings (
    ( antennaCreditRatio
      tx_layer
      'OXIDEn
      f_creditRatio
    )
) ; spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the minimum ratio between the sum of the credit provided by each protection device of a specified antenna model and the area of a specified metal layer.

Values

<i>tx_layer</i>	The metal layer to which the constraint applies.
<i>f_creditRatio</i>	The minimum ratio.

Parameters

'OXIDEn	The antenna model whose credit sum value is to be used. It can be one of 32 values ranging from OXIDE1 to OXIDE32. See also antennaModels .

Example

The minimum metal area to antenna credit ratio for Metal1 on OXIDE8 is specified as follows.

```
(antennaCreditRatio
  "Metal1"
  'OXIDE8
  100.5
)
```

antennaDiffusionWidthRatio

```
spacings (
    ( antennaDiffusionWidthRatio
        tx_layer
        ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]]
        'OXIDEn
        f_diffWidthRatio
    )
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the minimum ratio between the area of a specified metal layer and the sum of diffusion widths of all devices of a specified antenna model.

Values

tx_layer The metal layer to which the constraint applies.

f_diffWidthRatio The minimum ratio.

Parameters

'insideLayers (*tx_layer1 tx_layer2 ... tx_layerN*)

The constraint applies only if the shapes on *tx_layer* overlap a shape on one of these layers.

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

'insidePurposes (*t_purpose1 t_purpose2 ... t_purposeN*)

The *insideLayers* parameter is limited to the specified purposes.

Type: String (purpose name)

'OXIDEn

The antenna model to which the constraint applies. It can be one of 32 values ranging from OXIDE1 to OXIDE32.

See also [antennaModels](#).

Virtuoso Technology Data Constraints Reference

Antenna Constraints

Example

The minimum metal area to antenna diffusion width ratio for Metal1 on OXIDE1 is specified as follows. The ratio applies to Metal1 shapes connected to OXIDE1 type devices, identified by overlap with shapes on the LPP devTypeLayer, OXIDE1Purpose.

```
(antennaDiffusionWidthRatio
  "Metal1"
  'insideLayers ("devTypeLayer")
  'insidePurposes ("OXIDE1Purpose")
  'OXIDE1
  200.1
)
```

antennaModels

```
antennaModels (
  ( "t_name"
    t_constraintName(
      ( t_ratioName tx_layer f_value )
    )
  ...

  ( "factors"
    ( cumulativeRoutingPlusCut lx_layer )
    ( area lt_layer f_value )
    ( diffPlus lt_layer f_value )
    ( diffMinus lt_layer f_value )
    ( diffAreaReduce lt_layer ( ( f_area f_value ) ... )
  ) ;factors
) ;antennaModels
```

Defines antenna ratios for up to four oxide types, each specified in its own `antennaModels` group.

Antenna models must be specified only in the `antennaModels` constraint category of the foundry constraint group. The antenna models all follow the same syntax when specified in the ASCII technology file. For each antenna model, you can specify the `antenna`, `cumulativeMetalAntenna`, `cumulativeViaAntenna`, and `cumPerLayerAntenna` constraints, each of which specifies the `areaRatio` and `diffAreaRatio` ratios.

factors

Alternatively, you can specify `factors` by using the following syntax:

```
factors (
  ( cumulativeRoutingPlusCut lx_layer )
  ( area lt_layer f_value )
  ( diffPlus lt_layer f_value )
  ( diffMinus lt_layer f_value )
  ( diffAreaReduce lt_layer ( ( f_area f_value ) ... )
) ;factors
```

This is because `factors` is a SKILL technology class function and can be defined as `"name(...)"` or `"(name ...)"`.

Types

■ Add cut layer cumulative area ratio to metal cumulative area ratio

The calculation of the cumulative metal area ratio for a particular metal layer adds the partial cut area contribution from the cut layer below the layer in question as well as the partial metal area from the metal layer. Similarly, for the cumulative cut area ratio, the partial metal area for the metal layer below is added to the contribution from the partial cut area of the cut layer. The use of this antenna model is indicated by the `cumRoutingPlusCut` attribute on the layer.

$$Mi\_CAR = Mi-1\_CAR + Mi\_PAR + C(i, i-1)\_PAR$$

and

$$C(i, i-1)\_CAR = C(i-1, i-2)\_CAR + C(i, i-1)\_PAR + Mi-1\_PAR$$

where, `Mi_CAR` is the cumulative metal area ratio for metal layer `i`, `Mi_PAR` is the partial metal for the same metal layer, and `C(i, i-1)_PAR` is the partial cut area for the cut layer below `Mi`.

■ Subtract diffusion area from antenna area

The diffusion area is subtracted from the metal or cut area in the calculation of the antenna ratio for metal and cut layers.

$$\begin{aligned} \text{antenna\_ratio} &= (\text{metal\_area} - \text{diff\_minus\_factor} * \text{diff\_area}) / \text{gate\_area} \\ \text{antenna\_ratio} &= (\text{cut\_area} - \text{diff\_minus\_factor} * \text{diff\_area}) / \text{gate\_area} \end{aligned}$$

The diffusion area that is subtracted from the area is controlled by the `diff_minus_factor`. The default value of this factor is 0.

■ Add diffusion area to gate area

The diffusion area is added to the area of the gate in the calculation of the antenna ratio.

$$\text{antenna\_ratio} = \text{metal\_area} / (\text{gate\_area} + \text{diff\_plus\_factor} * \text{diff\_area})$$

Because the contribution from the diffusion area is taken into account in the antenna ratio, the antenna diff ratio (`metal_area/diff_area`) is not required to be calculated. This applies to both the layer antenna ratios and the cumulative antenna ratios for both metal and cut layers. The default value of `diff_plus_factor` is 0.

■ Reduce metal area with PWL diffusion factor

The metal area (and possibly cut area if the add via area of to the metal area model is applied) is multiplied by a factor called `diff_metal_reduce`, which depends on the diffusion area. The `diff_metal_reduce` factor applies to the `metal_area` before the contribution from `diff_minus_factor` is subtracted. This `diff_metal_reduce` factor is a piecewise linear model based on the diffusion area and typically ranges from

Virtuoso Technology Data Constraints Reference

Antenna Constraints

0 to 1. The default value is 1. A similar factor can be defined for cut and cumulative antenna ratios.

```
antenna_ratio = metal_area * diff_metal_reduce / gate_area
```

■ Area factor and side area factor

The default value for `area_factor` is 1. Because the antenna ratio can either be side or non-side at one time, there is a need to store only one `area_factor`, either the side or non-side factor. A Boolean operator indicates whether the factor is used for the side area factor.

```
antenna_ratio = area_factor * metal_area / gate_area
```

or

```
antenna_side_ratio = area_factor * metal_side_area / gate_area
```

■ Combined antenna models

Adding all possible factors, the antenna ratio for a metal layer can be calculated as follows:

```
antenna_ratio = (area_factor * metal_area * diff_reduce_factorPWL -  
diff_minus_factor * diff_area) / (gate_area + diff_plus_factor * diff_area)
```

For a cut layer, the antenna ratio is calculated as follows:

```
antenna_ratio = (area_factor * cut_area * diff_reduce_factorPWL -  
diff_minus_factor * diff_area) / (gate_area + diff_plus_factor * diff_area)
```

These ratios are associated with a particular layer. However, when performing cumulative antenna calculations, the appropriate layer antenna factors should be used. These are not associated with particular oxide models.

Virtuoso Technology Data Constraints Reference

Antenna Constraints

Parameters

<i>t_name</i>	The antenna oxide model name. Valid values are as follows: ■ <code>default</code>: Antenna oxide1 model ■ <code>second</code>: Antenna oxide2 model ■ <code>third</code>: Antenna oxide3 model ■ <code>fourth</code>: Antenna oxide4 model
<i>t_constraintName</i>	The name of the constraint being defined. Valid values are as follows: ■ <code>antenna</code> specifies the maximum antenna ratios for one or more antenna models for the specified layer. ■ <code>cumulativeMetalAntenna</code> specifies the maximum antenna ratios for one or more antenna models for area values accumulated through all the poly and metal layers attached to a gate. ■ <code>cumulativeViaAntenna</code> specifies the maximum antenna ratios for one or more antenna models for area values accumulated through all the cut and via layers attached to a gate. ■ <code>cumPerLayerAntenna</code> specifies the maximum antenna ratios for one or more antenna models for area values accumulated for the specified layer.
<i>t_ratioName</i>	The name of the antenna ratio, <code>areaRatio</code> or <code>diffAreaRatio</code>.
<i>tx_layer</i>	The layer on which the antenna ratio constraint is applied. It must not be specified when <i>t_constraintName</i> is <code>cumulativeMetalAntenna</code> or <code>cumulativeViaAntenna</code>. Type: String (layer name) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Antenna Constraints

f_value

The *areaRatio* or *diffAreaRatio* value.

- *areaRatio* has the following syntax:

f_value ['side]

where,

- *f_value* is the area ratio
- 'side specifies that the side calculation be applied.

The default is "no side".

- *diffAreaRatio* has the following syntax:

f_value | ((*f_area* *f_value*)...) ['side]

where,

- *f_value* is the diffusion area ratio, specified as an absolute value or based on the total diffusion area, if specified.
- *f_area* is the total diffusion area.
- 'side specifies that the side calculation be applied.

The default is "no side".

Example

```
antennaModels(  
  ;( model  
  ( "default"  
    antenna(  
      ;(ratioName layer value)  
      ( areaRatio METAL1 450.0 )  
      ( diffAreaRatio METAL1 ((0.45 0.4) (0.55 0.6)) )  
      ( areaRatio VIA1 0.344 )  
      ( diffAreaRatio VIA1 ((0.44 0.5) (0.57 0.8)) )  
    ) ;antenna  
    cumulativeMetalAntenna(  
      ;( ratioName value )  
      ( areaRatio 0.544 'side )  
      ( diffAreaRatio 0.888 )  
    ) ;cumulativeMetalAntenna  
    cumulativeViaAntenna(  
      ;( ratioName value )  
      ( areaRatio 0.654 'side )  
      ( diffAreaRatio 0.884 )  
    ) ;cumulativeViaAntenna  
  ) ;default
```

Virtuoso Technology Data Constraints Reference

Antenna Constraints

```
( "second"
  antenna(
    ;( ratioName layer value )
    ( areaRatio METAL1 1.345 'side )
    ( diffAreaRatio METAL1 ((0.455 0.4) (0.555 0.65)) )
  ) ;antenna
  cumulativeMetalAntenna(
    ;( ratioName value )
    ( areaRatio (3.564 nil) )
    ( diffAreaRatio 7.888 )
  ) ;cumulativeMetalAntenna
  cumulativeViaAntenna(
    ;( ratioName value )
    ( areaRatio (7.654 nil) )
    ( diffAreaRatio 1.884 )
  ) ;cumulativeViaAntenna
) ;second

( "third"
  antenna(
    ;( ratioName layer value )
    ( areaRatio METAL1 0.343 )
    ( diffAreaRatio METAL1 ((0.43 0.3) (0.53 0.6)) )
  ) ;antenna
  cumulativeMetalAntenna(
    ;( ratioName value )
    ( areaRatio 9.765 'side )
    ( diffAreaRatio 0.188 )
  ) ;cumulativeMetalAntenna
  cumulativeViaAntenna(
    ;( ratioName value )
    ( areaRatio 3.22 )
    ( diffAreaRatio ((0.421 0.4) (0.7 0.64) (0.81 0.59)) )
  ) ;cumulativeViaAntenna
) ;third

( "fourth"
  antenna(
    ;( ratioName layer value )
    ( areaRatio METAL2 0.645 'side )
    ( diffAreaRatio METAL2 ((0.47 0.4) (0.55 0.7)) )
  ) ;antenna
  cumulativeMetalAntenna(
    ;( ratioName value )
    ( areaRatio 0.6 )
    ( diffAreaRatio ((0.127 0.23) (0.32 0.7)) )
  ) ;cumulativeMetalAntenna
  cumulativeViaAntenna(
    ;( ratioName value )
    ( areaRatio 9.67 )
    ( diffAreaRatio 0.684 )
  ) ;cumulativeViaAntenna
) ;fourth
) ;antennaModels

cumPerLayerAntenna(
  ( areaRatio "via1" 0.64 'side )
  ( diffAreaRatio "via1" ((30 0.3888) (25 0.6774)) 'side )
) ;cumPerLayerAntenna
```

antennaModelCombo

```
antennaModelCombo (  
    'and (t_modelName1 ...) t_modelName  
) ;antennaModelCombo
```

(Virtuoso Advanced Node for Layout Only) Defines the antenna oxide model to use for nets that are connected to devices with at least one each of a list of other antenna oxide models.

Values

<i>t_modelName</i>	The name of the antenna oxide model to be used.
	The valid values are OXIDE1, OXIDE2, OXIDE3, and OXIDE4.

Parameters

<code>'and (t_modelName1 ...)</code>	The antenna oxide models that should be part of the <code>and</code> operation.

Example

The antenna oxide model OXIDE3 is specified as the one to use for nets that contain devices with at least one each of the antenna oxide models OXIDE1 and OXIDE2.

```
spacings (  
    (antennaModelCombo 'and ("OXIDE1" "OXIDE2" ) "OXIDE3")  
) ;spacings
```

antennaRatio

```

spacings (
    ( antennaRatio
        tx_layer
        'OXIDEn
        ['isSide]
        ['diffAreaRatio g_table]
        ['areaFactor f_areaFactor]
        ['diffuseOnly]
        [['diffPlusFactor f_diffPlusFactor]
          | ['gatePlusDiffPWL g_gatePlusDiffTable]
        ]
        ['diffMinusFactor f_diffMinusFactor]
        ['diffAreaReduceFactor g_table]
        ['gatePWL g_gatePWLTable]
        f_areaRatio
    )
) ;spacings

```

Defines antenna ratios on a per-layer basis. Different ratios can be specified for different types of devices, described as different OXIDE models. For example, a transistor with a thinner oxide would have different antenna ratios than transistors with thicker oxides. The thinner oxide transistor could be modeled as OXIDE1 and the thicker oxide transistor could be modeled as OXIDE2. The OXIDE1 transistor area ratio would be smaller than the OXIDE2 transistor area ratio because the thinner oxide is more easily damaged. For detailed information about antenna model constraints, see [antennaModels](#).

When `gatePWL` and either `diffPlusFactor` or `gatePlusDiffPWL` are specified, the gate area used to calculate `f_areaRatio` is determined using the following formula:
`gatePWL(gate) + plusDiffProtect(diff)`

Here:

- `gatePWL(gate)` indicates the gate area defined using `gatePWL`.
- `plusDiffProtect(diff)` indicates the diffusion area that is added to the gate area value.

If `diffPlusFactor` is specified, then:

`plusDiffProtect = plusDiffFactor * diffArea`

If `gatePlusDiffPWL` is specified, then:

`plusDiffProtect = interpolated PWL value indexed by diffArea`

Virtuoso Technology Data Constraints Reference

Antenna Constraints

Values

<i>tx_layer</i>	The metal layer to which the constraint applies.
<i>f_areaRatio</i>	The area ratio of the metal area to the connected gate area.

Parameters

'OXIDEn	Indicates the antenna model to which the constraint applies. It can be one of 32 values ranging from OXIDE1 to OXIDE32.
'isSide	By default, the antenna ratio specifies the ratio of the metal area to the gate area. If <i>isSide</i> is specified, it indicates that the ratio is measured from the metal side area as follows: $(\text{perimeter} \times \text{thickness}) / \text{gate area}$
'diffAreaRatio <i>g_table</i>	The <i>g_table</i> row is in this format: <pre>(("area")) (<i>f_diffArea1</i> <i>f_metalToDiffAreaRatio1</i>) (<i>f_diffArea2</i> <i>f_metalToDiffAreaRatio2</i>) ...)</pre> The index <i>f_diffArea</i> is the diffusion area, and the table value <i>f_metalToDiffAreaRatio</i> is the required metal area-to-diffusion area ratio. The table represents a PWL model, in which the <i>diffAreaRatio</i> ratio is calculated using a linear interpolation of the diffusion areas. Type: A 1-D table indexed by a float value.
'areaFactor <i>f_areaFactor</i>	Area ratios can be raised or lowered by the specified <i>f_areaFactor</i> value. When checking antenna rules, partial area ratios are calculated by multiplying the metal or gate ratio by <i>f_areaFactor</i> . These calculated ratios are stored on abstract pins.
'diffuseOnly	Indicates that the <i>f_areaFactor</i> value is applied only to partial area ratios for the metal attached to the diffusion.

Virtuoso Technology Data Constraints Reference

Antenna Constraints

```
['diffPlusFactor f_diffPlusFactor]  
| ['gatePlusDiffPWL g_gatePlusDiffTable]
```

Indicates the diffusion area that is added to the gate area value. By default, the metal to gate ratio is measured.

If a *f_diffPlusFactor* value is specified, the gate area used to calculate the ratio is adjusted by adding the value obtained by multiplying *f_diffPlusFactor* by *f_diffAreaRatio*.

Instead, values can be specified separately for each oxide in the *g_gatePlusDiffTable* 1-D table. Each row in the table is in this format:

```
( ("area") )  
    ( f_diffArea1 f_plusDiffProtect1 )  
    ( f_diffArea2 f_plusDiffProtect2 ) ...  
)
```

The index *f_diffArea* is the diffusion area, and the table value *f_plusDiffProtect* is the additional protection provided by the diffusion area to the gate area.

For example:

```
'gatePlusDiffPWL ((( "area" ))  
(  
    0.45 0.03  
    1.1  0.1 )  
)  
)
```

```
'diffMinusFactor f_diffMinusFactor
```

By default, the metal to gate ratio is measured. If an *f_diffMinusFactor* value is specified, the metal area used to calculate the ratio is adjusted by subtracting the value obtained by multiplying *f_diffMinusFactor* by *f_diffAreaRatio*.

Virtuoso Technology Data Constraints Reference

Antenna Constraints

'diffAreaReduceFactor *g_table*

The *g_table* row is in this format:

```
( ("area") )  
  ( f_diffArea1 f_diffAreaReduceFactor1 )  
  ( f_diffArea2 f_diffAreaReduceFactor2 ) ...  
)
```

The index *f_diffArea* is the diffusion area, and the table value *f_diffAreaReduceFactor* is the factor used to adjust the diffusion area. The diffusion area used to calculate the ratio is adjusted by multiplying it by the *diffAreaReduceFactor* calculated from the table. The table represents a piecewise linear model, in which the required *diffAreaReduceFactor* value is calculated using a linear interpolation of the diffusion areas.

Type: A 1-D table indexed by a float value.

'gatePWL *g_gatePWLTable*

Specifies a PWL (piece-wise linear) table on the given antenna model that is indexed by the real gate area, and returns an effective gate area interpolated from the table. Each row in the table is in this format:

```
( ("area") )  
  ( f_gateArea1 f_effectiveGateArea1 )  
  ( f_gateArea2 f_effectiveGateArea2 ) ...  
)
```

The index *f_gateArea* is the real gate area, and the table value *f_effectiveGateArea* is the effective gate area interpolated from the PWL table.

For example:

```
'gatePlusDiffPWL ((( "area" ) )  
(  
  0.45 0.03  
  1.1  0.1 )  
)  
)
```

Type: A 1-D table indexed by a float value.

Virtuoso Technology Data Constraints Reference

Antenna Constraints

Example

The antenna ratio for VIA1 is specified as follows.

```
spacings(  
  ;OXIDE3 rules  
  ( antennaRatio "VIA1" 'OXIDE3  
    'diffAreaRatio  
      ( ("area") )  
      (  
        0.0 100.0  
      )  
    )  
    'diffPlusFactor 5.0  
    'diffAreaReduceFactor  
      ( ("area") )  
      (  
        0.0 0.8  
      )  
    )  
    100.0  
  )  
)
```

cumPerLayerAntennaRatio

```
spacings (
    ( cumPerLayerAntennaRatio
        tx_layer
        'OXIDEn
        ['isSide]
        ['diffAreaRatio g_table]
        ['areaFactor f_areaFactor]
        ['diffuseOnly]
        [['diffPlusFactor f_diffPlusFactor]
        | ['gatePlusDiffPWL g_gatePlusDiffTable]
        ]
        ['diffMinusFactor f_diffMinusFactor]
        ['diffAreaReduceFactor g_table]
        ['gatePWL g_gatePWLTable]
        ['cumRoutingPlusCut]
        f_areaRatio
    )
) ;spacings
```

Defines cumulative antenna ratios on a per-layer basis. This constraint adds all partial area ratios measured for each net in each layer below the specified layer and flags those that exceed the specified permissible ratios. Unique constraints can be specified for each of the 32 permissible OXIDE models. For detailed information about antenna model constraints, see [antennaModels](#).

When `gatePWL` and either `diffPlusFactor` or `gatePlusDiffPWL` are specified, the gate area used to calculate `f_areaRatio` is determined using the following formula:
`gatePWL(gate) + plusDiffProtect(diff)`

Here:

- `gatePWL(gate)` indicates the gate area defined using `gatePWL`.
- `plusDiffProtect(diff)` indicates the diffusion area that is added to the gate area value.

If `diffPlusFactor` is specified, then:

`plusDiffProtect = plusDiffFactor * diffArea`

If `gatePlusDiffPWL` is specified, then:

`plusDiffProtect = interpolated PWL value indexed by diffArea`

Virtuoso Technology Data Constraints Reference

Antenna Constraints

Values

<i>tx_layer</i>	The metal layer to which the constraint applies.
<i>f_areaRatio</i>	The area ratio of the metal area to the connected gate area.

Parameters

'OXIDEn	Indicates the antenna model to which the constraint applies. It can be one of 32 values ranging from OXIDE1 to OXIDE32.
'isSide	By default, the antenna ratio specifies the ratio of the metal area to the gate area. If <i>isSide</i> is specified, it indicates that the ratio is measured from the metal side area as follows: (perimeter x thickness) / gate area
'diffAreaRatio <i>g_table</i>	The <i>g_table</i> row is in this format: <pre>(("area")) (<i>f_diffArea1</i> <i>f_metalToDiffAreaRatio1</i>) (<i>f_diffArea2</i> <i>f_metalToDiffAreaRatio2</i>) ...)</pre> The index <i>f_diffArea</i> is the diffusion area, and the table value <i>f_metalToDiffAreaRatio</i> is the required metal area-to-diffusion area ratio. The table represents a piecewise linear model, in which the <i>diffAreaRatio</i> ratio is calculated using a linear interpolation of the diffusion areas. Type: A 1-D table indexed by a float value.
'areaFactor <i>f_areaFactor</i>	Area ratios can be raised or lowered by the specified <i>f_areaFactor</i> value. When checking antenna rules, partial area ratios are calculated by multiplying the metal or gate ratio by <i>f_areaFactor</i> . These calculated ratios are stored on abstract pins.
'diffuseOnly	Indicates that the <i>f_areaFactor</i> value is applied only to partial area ratios for the metal attached to the diffusion.

Virtuoso Technology Data Constraints Reference

Antenna Constraints

```
['diffPlusFactor f_diffPlusFactor]  
| ['gatePlusDiffPWL g_gatePlusDiffTable]
```

Indicates the diffusion area that is added to the gate area value. By default, the metal to gate ratio is measured.

If a *f_diffPlusFactor* value is specified, the gate area used to calculate the ratio is adjusted by adding the value obtained by multiplying *f_diffPlusFactor* by *f_diffAreaRatio*.

Instead, values can be specified separately for each oxide in the *g_gatePlusDiffTable* 1-D table. Each row in the table is in this format:

```
( ("area") )  
    ( f_diffArea1 f_plusDiffProtect1 )  
    ( f_diffArea2 f_plusDiffProtect2 ) ...  
)
```

The index *f_diffArea* is the diffusion area, and the table value *f_plusDiffProtect* is the additional protection provided by the diffusion area to the gate area.

For example:

```
'gatePlusDiffPWL ((( "area" ))  
(  
    0.45 0.03  
    1.1  0.1 )  
)  
)
```

```
'diffMinusFactor f_diffMinusFactor
```

By default, the metal or gate ratio is measured. If an *f_diffMinusFactor* value is specified, the metal area used to calculate the ratio is adjusted by subtracting the value obtained by multiplying *f_diffMinusFactor* by *f_diffAreaRatio*.

'diffAreaReduceFactor *g_table*

The *g_table* row is in this format:

```
( ( "area" )
  ( f_diffArea1 f_diffAreaReduceFactor1 )
  ( f_diffArea2 f_diffAreaReduceFactor2 ) ...
)
```

The index *f_diffArea* is the diffusion area, and the table value *f_diffAreaReduceFactor* is the factor used to adjust the diffusion area. The diffusion area used to calculate the ratio is adjusted by multiplying it by the *diffAreaReduceFactor* calculated from the table. The table represents a piecewise linear model, in which the required *diffAreaReduceFactor* value is calculated using a linear interpolation of the diffusion areas.

Type: A 1-D table indexed by a float value.

'gatePWL *g_gatePWTable*

Specifies a PWL (piece-wise linear) table on the given antenna model that is indexed by the real gate area, and returns an effective gate area interpolated from the table. Each row in the table is in this format:

```
( ( "area" )
  ( f_gateArea1 f_effectiveGateArea1 )
  ( f_gateArea2 f_effectiveGateArea2 ) ...
)
```

The index *f_gateArea* is the real gate area, and the table value *f_effectiveGateArea* is the effective gate area interpolated from the PWL table.

For example:

```
'gatePlusDiffPWL ((( "area" ))
(
  0.45 0.03
  1.1  0.1 )
)
```

Type: A 1-D table indexed by a float value.

'cumRoutingPlusCut

By default, this constraint includes the partial area ratios from the metal layer below. If *cumRoutingPlusCut* is specified, the cumulative antenna ratio of the cut layer immediately below is included for the cumulative antenna calculation for the specified layer.

Virtuoso Technology Data Constraints Reference

Antenna Constraints

Example

The cumulative antenna ratio for VIA1 is specified as follows.

```
spacings (
  ;OXIDE3 rules
  ( cumPerLayerAntennaRatio "VIA1" 'OXIDE3
    'isSide
    'diffAreaRatio
      ( ("area") )
      (
        0.0 100.0
      )
    )
    'areaFactor 0.4
    'diffuseOnly
    'diffPlusFactor 0.5
    'diffMinusFactor 0.6
    'diffAreaReduceFactor
      ( ("area") )
      (
        0.0 0.8
      )
    )
    'cumRoutingPlusCut
    100.0
  )
)
```

diffWidthPlusFactor

```
spacingTables(
  ( diffWidthPlusFactor
    (( "stack" nil nil )
     ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]]
    ]
    'connectLayer tx_connectLayer
  )
  ['backside]
  g_table
)
) ;spacingTables
```

(Virtuoso Layout Suite EXL and higher tiers) Defines the factor used during antenna extraction by Abstract Generator when calculating the amount of protection that a protection device provides.

Values

"stack" nil nil Identifies the index for the table.

g_table The format of the *table* row is as follows:

(f_stack) f_factor

or

f_stack f_factor

where,

- *f_stack* is the device stack. The stack is determined by the distance of the gate to the connection layer.
- *f_factor* is the amount of protection each stacked device provides to a net.

Type: A 1-D table specifying integer stack and floating-point *f_factor* values.

Parameters

'insideLayers (*tx_layer1 tx_layer2 ... tx_layerN*)

Virtuoso Technology Data Constraints Reference

Antenna Constraints

The constraint applies only to the specified layers of the protection device.

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `insideLayers` parameter is limited to the specified purposes.

Type: String (purpose name)

`'connectLayer tx_connectLayer`

The layer to which the protection device is to be connected to provide protection.

`'backside`

The constraint defines the protection factor associated with diffusion width. If `backside` is specified, the factor applies when calculating protection for backside metal layers. If not specified, the factor applies when calculating protection for frontside metal layers. For a process with both frontside and backside layers, two constraints must be defined, one with the `backside` parameter and one without it.

Example

Defines the credit factors for a net. Both constraints specify the connect layer as Metal2 and credit factors indexed by the device stack number. The second constraint additionally defines layers and purposes and that the credit factors need to be considered for the metal backside.

```
spacingTables (
  (diffWidthPlusFactor
    (( "stack" nil nil )
      'connectLayer "Metal2"
    )
    (
      1 1000.0
      2 1700.0
    )
  )
  (diffWidthPlusFactor
    (( "stack" nil nil )
      'insideLayers ("Metal2")
      'insidePurposes ("fill" "pin")
      'connectLayer "Metal2"
      'backside)
    (
      1 2000.0
      2 1800.0
    )
  )
); spacingTables
```

maxFloatingArea

```

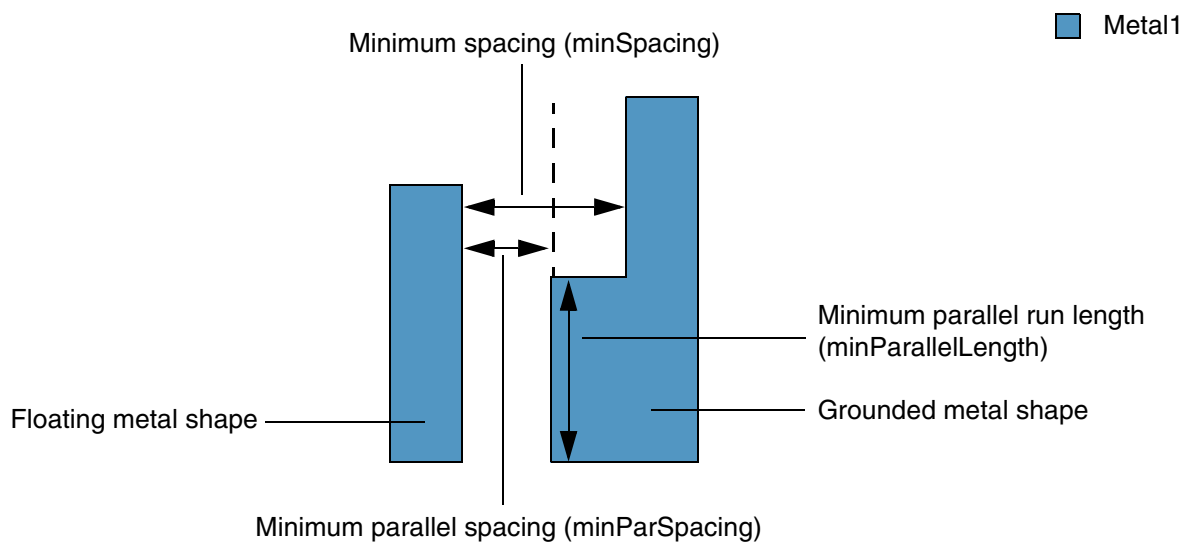
spacings (
  ( maxFloatingArea
    'routingLayers (tx_layer1 tx_layer2 ... tx_layerN)
    'singleLayer | 'connected | 'allConnected
    'spacings
    (
      (( "layer" ))
      (g_table)
    )
    f_area
  )
) ;spacings

```

Defines the maximum allowed area for floating metal shapes on the specified routing layers.

Floating metal shapes are shapes on the current layer that cannot trace a path to a diffusion connection or polysilicon gate by using only same-layer or lower-layer connections. Grounded metal shapes are those that can connect to a diffusion connection or polysilicon gate by using same-layer or lower-layer connections.

Multiple `maxFloatingArea` constraints must be placed in an `'or` constraint group.



Virtuoso Technology Data Constraints Reference

Antenna Constraints

Values

f_area The area of floating metal shapes must be less than or equal to this value.

Parameters

`'routingLayers (tx_layer1 tx_layer2 ... tx_layerN)`

List of routing layers to which the constraint applies.

Type: String (layer name) or Integer (layer number)

`'singleLayer | 'connected | 'allConnected`

The constraint can be applied as follows:

- `'singleLayer`: The constraint applies to the area of each floating metal shape on each specified routing layer. Each shape must have an area that is less than or equal to *area*, or meet the minimum spacing requirements between floating and grounded metal shapes.
- `'connected`: The constraint applies to the sum of the areas of the connected floating metal shapes on each specified routing layer. The connected area on a layer must be less than or equal to *area*, or meet the minimum spacing requirements between the floating and grounded metal shapes.
- `'allConnected`: The constraint applies to the sum of the areas of all connected floating metal shapes on the specified routing layers. The total connected area on the current and lower layers must be less than or equal to *area*, or meet the minimum spacing requirements between the floating and grounded metal shapes.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Antenna Constraints

```
'spacings( (( "layer" )) (g_table) )
```

"layer" identifies the index for table.

Each entry in *g_table* takes the following form:

```
tx_layer (
  (f_minSpacing 0.0)
  (f_minParSpacing f_parallelLength)
  ...
)
```

where, *minSpacing* is the minimum spacing, *minParSpacing* is the minimum parallel spacing, and *parallelLength* is the minimum parallel run length, as shown in the figure above.

If the floating metal area for the current layer (and optionally for all lower connected layers) is greater than *area*, one of the following conditions must be met:

- The distance between the floating and grounded shapes must be greater than or equal to the specified distance (*minSpacing*).
- The floating metal shapes that are at a distance greater than or equal to *minParSpacing* from grounded metal shapes must have parallel run length greater than or equal to *parallelLength* at the minimum spacing distance.

The *minParSpacing* values must be defined in the decreasing order and must be smaller than the *minSpacing* values. Smaller *minParSpacing* spacing values require larger *parallelLength* values.

Type: A 1-D table indexed on "layer", specifying floating-point spacing and parallel run length values that apply to the shapes on each specified routing layer

Virtuoso Technology Data Constraints Reference

Antenna Constraints

Example

If the sum of areas of floating metal shapes is greater than 1000, the following conditions must be met:

- For layer Metal1, the distance between a floating metal shape and a grounded metal shape must be greater than or equal to 1.0; otherwise,
 - If the distance between them is greater than or equal to 0.5, the parallel run length at minimum spacing must be at least 0.8.
 - If the distance between them is greater than or equal to 0.2, the parallel run length at minimum spacing must be at least 2.0. Spacing that is less than 0.2 is not allowed.
- For Metal2 through Metal5, the distance between a floating metal shape and a grounded metal shape must be greater than or equal to 0.6; otherwise, if the distance between them is greater than or equal to 0.3, the parallel run length at the minimum spacing must be at least 0.8.

Note: The layers the constraint is defined with must match the entries in the table. The first value pair corresponds to the spacing value with zero parallel run length. The other value pairs specify the corresponding parallel spacing and parallel run length values.

Virtuoso Technology Data Constraints Reference

Antenna Constraints

```

spacings (
  ( maxFloatingArea
    'routingLayers ( "Metal1" "Metal2" "Metal3" "Metal4" "Metal5" )
    'connected
    'spacings ( ( ("layer"))
      ("Metal1" ((1.0 0.0 ) (0.5 0.8) (0.2 2.0 ) )
      "Metal2" ((0.6 0.0 ) (0.3 0.8))
      "Metal3" ((0.6 0.0 ) (0.3 0.8))
      "Metal4" ((0.6 0.0 ) (0.3 0.8))
      "Metal5" ((0.6 0.0 ) (0.3 0.8))
    )
  )
  1000.0
) ;spacings

```

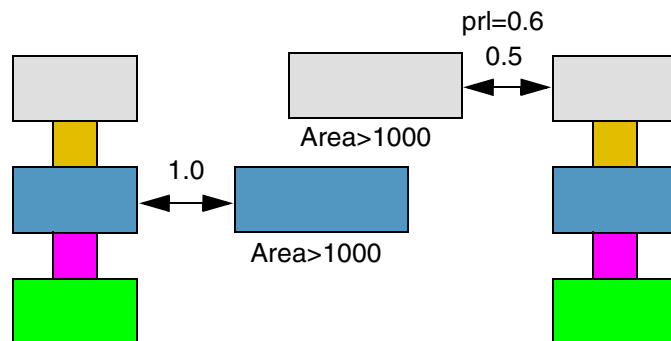
Metal2

Via1

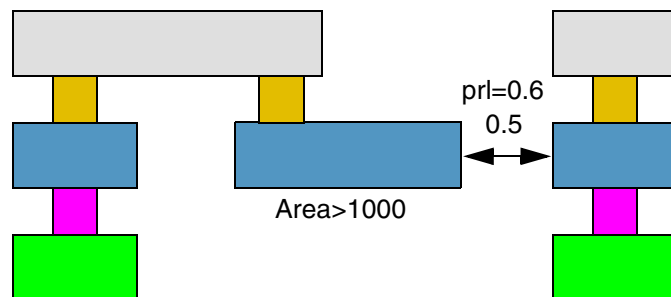
Metal1

CO

Gate



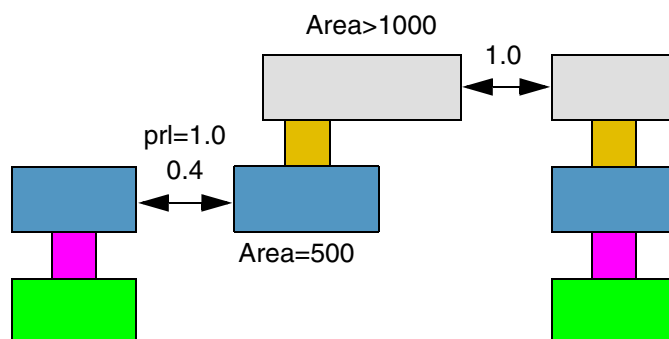
a) FAIL. The spacing between the floating and grounded Metal1 shapes is 1.0, which satisfies the *minSpacing* requirement. The spacing between the floating and grounded Metal2 shapes is 0.5, which satisfies *minParSpacing*, but the parallel run length (prl) between them is only 0.6 (<0.8).



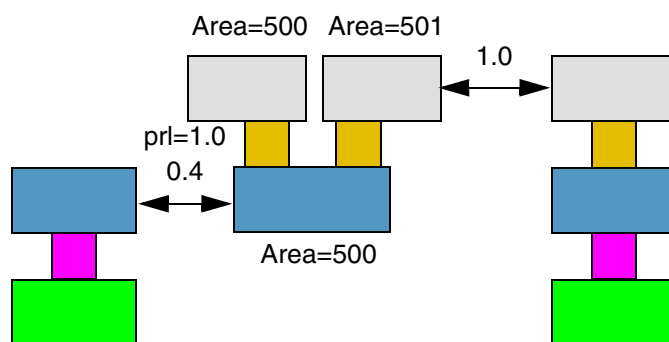
b) FAIL. When Metal1 is fabricated the large Metal1 shape is floating and is 0.5 away from the grounded Metal1 shape. This requires a prl of 0.08, but the prl is only 0.6.

Virtuoso Technology Data Constraints Reference

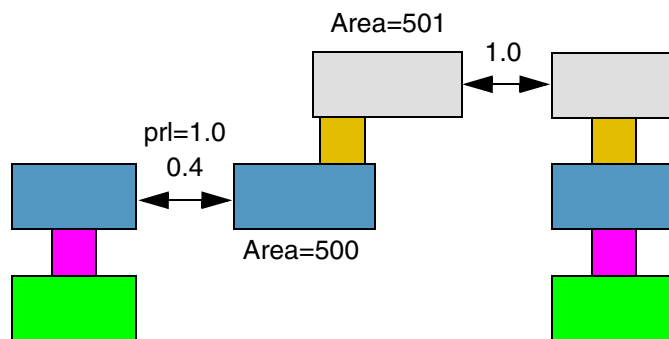
Antenna Constraints



c) FAIL. The floating Metal2 shape satisfies the *minSpacing* requirement, but is connected to a floating Metal1 shape that does meet the *prl* requirement of 2.0 (required when *minParSpacing* (0.4) is greater than or equal to 0.2).



d) FAIL. The total connected Metal2 area is > 1000 (500 + 501) and satisfies the *minSpacing* requirement. However, the two Metal2 shapes are connected to a Metal1 shape that does meet the *prl* requirement of 2.0 (required when *minParSpacing* (0.4) is greater than or equal to 0.2).



e) The constraint does not apply because the floating Metal1 and Metal2 shapes have area less than 1000. A violation would occur if 'allConnected' was specified.

Area Constraints

This topic includes the following constraints:

- maxCumAntenna
- minArea
- minAreaEdgeLength
- minHoleArea
- minIsolatedArea
- minRectArea

Area values are stored after multiplying them with DBU-square (DBU*DBU).

maxCumAntenna

```
spacings (
    ( maxCumAntenna
      ['lowerLayer tx_lowerLayer]
      ['upperLayer tx_upperLayer]
      ['cutLayerOnly]
      f_area )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the maximum cumulative area of all layers that are not connected to the diffusion diode. Optionally, the constraint applies only to a layer range and not to all layers.

Values

<i>f_area</i>	The maximum cumulative area of all layers must be less than or equal to this value.
	Type: Float value with area units

Parameters

<i>'lowerLayer tx_lowerLayer</i>	(Virtuoso Layout Suite EXL and higher tiers) The maximum cumulative area is between this layer to the layer specified with <i>upperLayer</i> .
<i>'upperLayer tx_upperLayer</i>	(Virtuoso Layout Suite EXL and higher tiers) The maximum cumulative area is between the layer specified with <i>lowerLayer</i> to this layer.
<i>'cutLayerOnly</i>	(Virtuoso Layout Suite EXL and higher tiers) Only cut layers are considered when calculating the cumulative area.

Example

The cumulative area of all layers from Metal4 to Metal 5 must be less than or equal to 20000.0.

```
spacings (
    ( maxCumAntenna
```

Virtuoso Technology Data Constraints Reference

Area Constraints

```
        'lowerLayer "Metal4"  
        'upperLayer "Metal5"  
        20000.0  
    )  
); spacings
```

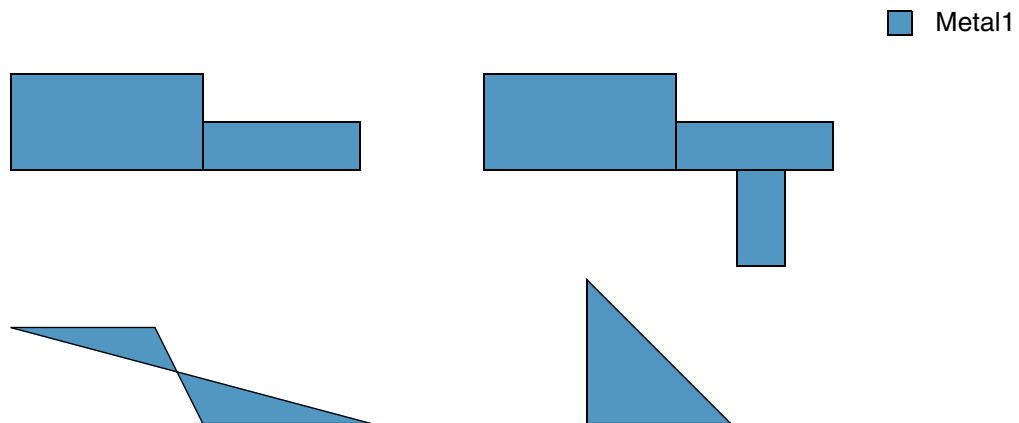
minArea

```
spacings(  
  ( minArea tx_layer  
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)']  
      |  
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)']  
    ]  
    f_area  
  )  
) ;spacings
```

Specifies the minimum metal area required for non-rectangular polygon shapes on a layer. Minimum area rules most often apply to via contacts, especially for stacked vias.

To specify the minimum area allowed for rectangular shapes on a layer, use the [minRectArea](#) constraint. If a [minRectArea](#) constraint is not specified on that layer, the [minArea](#) constraint applies to all shapes.

Note: If [minArea](#) and [minSize](#) are set on a layer, both constraints are satisfied if either constraint is met.



The area of a shape on the specified layer must be greater than or equal to the value specified by the constraint.

Virtuoso Technology Data Constraints Reference

Area Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_area</i>	The area of a shape on the specified layer must be greater than or equal to this value. Type: Float value with area units

Parameters

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Area Constraints

Example

The minimum area of a Metal1 shape must be greater than or equal to 5.0.

```
spacings(  
    ( minArea "Metal1"  
      5.0  
    )  
); spacings
```

minAreaEdgeLength

```
spacings (
    ( minAreaEdgeLength tx_layer
      [['exceptMinWidth f_width]
      |
      [['exceptEdgeLength f_edgeLength
        | (f_minEdgeLength f_maxEdgeLength)]
      ['exceptMinSize (f_minWidth f_minLength)
        | ((f_minWidth f_minLength) ...)]
      ['exceptStep (f_length1 f_length2)]
      ]
    ]
    f_area
  )
) ;spacings
```

Specifies the minimum area for a polygon if all of its edges are less than a given length, or if a rectangle with dimensions *minWidth* and *minLength* cannot fit inside the polygon.

Optionally, the constraint does not apply if a polygon has two adjacent edges, one of which is greater than or equal to *length1* and the other is less than *length2* in length.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_area</i>	The area of the polygon must be greater than or equal to this value. Type: Float value with area units

Virtuoso Technology Data Constraints Reference

Area Constraints

Parameters

`'exceptMinWidth f_width`

The constraint does not apply if any width of the polygon is greater than or equal to this value.

`'exceptEdgeLength f_edgeLength`
`| (f_minEdgeLength f_maxEdgeLength)`

The constraint does not apply if the length of at least one edge of the polygon is greater than or equal to *edgeLength*.

If both *minEdgeLength* and *maxEdgeLength* are specified instead, the constraint applies only if the polygon has at least one edge with length greater than or equal to *minEdgeLength* and less than *maxEdgeLength*.

`'exceptMinSize (f_minWidth f_minLength)`
`| ((f_minWidth f_minLength)...)`

The constraint does not apply if a rectangle with this width and length can fit inside the polygon.

When multiple width and length pairs are specified, the constraint does not apply if any one of the given rectangles fits inside the polygon.

`'exceptStep (f_length1 f_length2)`

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if an edge with length greater than or equal to *length1* has an adjacent edge with length less than *length2*.

Examples

- Example 1: minAreaEdgeLength with exceptEdgeLength
- Example 2: minAreaEdgeLength with exceptMinSize
- Example 3: minAreaEdgeLength with exceptMinWidth
- Example 4: minAreaEdgeLength with exceptStep

Virtuoso Technology Data Constraints Reference

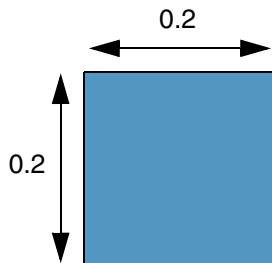
Area Constraints

Example 1: *minAreaEdgeLength* with *exceptEdgeLength*

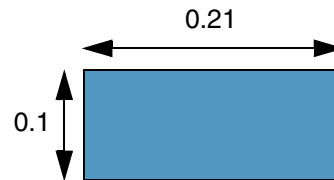
The area of a polygon must be at least 0.04 if all its edges are less than 0.21 long.

```
spacings(  
  ( minAreaEdgeLength "Metal1"  
    'exceptEdgeLength 0.21  
    0.04  
  )  
) ;spacings
```

■ Metal1



a) PASS. The length of the edges is 0.2 (< 0.21) and the area of the shape is 0.04.



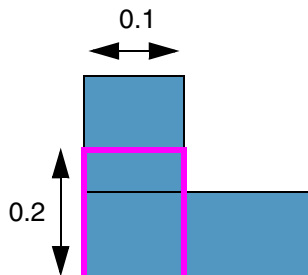
b) The constraint does not apply because one edge is 0.21 long.

Example 2: *minAreaEdgeLength* with *exceptMinSize*

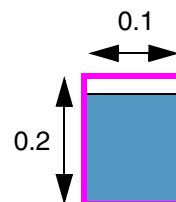
The area of a polygon must be at least 0.6 if a 0.1x0.2 rectangle cannot fit inside it.

```
spacings(  
  ( minAreaEdgeLength "Metal1"  
    'exceptMinSize (0.1 0.2)  
    0.6  
  )  
) ;spacings
```

■ Metal1



a) The constraint does not apply because a 0.1x0.2 rectangle fits inside the polygon.



b) FAIL. The 0.1x0.2 rectangle does not fit inside the polygon. Therefore, the area of the polygon must be at least 0.6.

Virtuoso Technology Data Constraints Reference

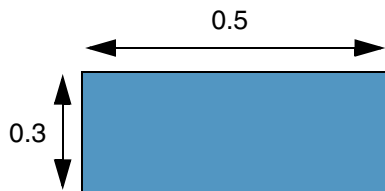
Area Constraints

Example 3: *minAreaEdgeLength* with *exceptMinWidth*

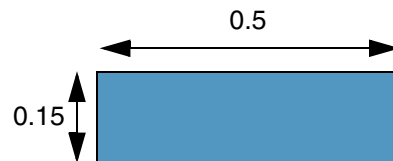
The area of a polygon must be at least 0.18 if the width of the shape is less than 0.2.

```
spacings (
  ( minAreaEdgeLength "Metal1"
    'exceptMinWidth 0.2
    0.18
  )
) ;spacings
```

 Metal1



a) The constraint does not apply because the width of the polygon is greater than 0.2.



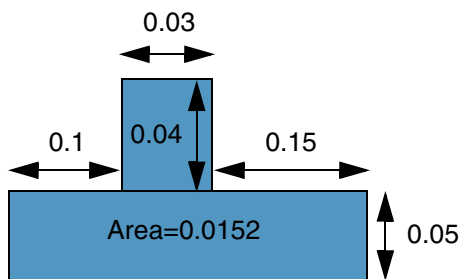
b) FAIL. The area of the polygon is 0.075 (<0.18).

Example 4: minAreaEdgeLength with exceptStep

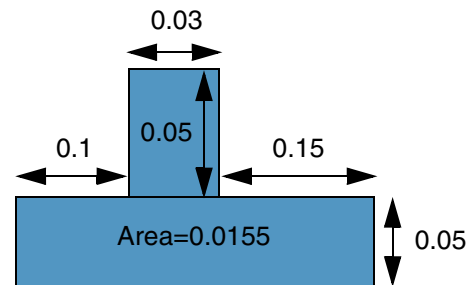
The area of a polygon must be at least 0.017 if it has two adjacent edges, one of which is greater than or equal to 0.12 and the other is less than 0.05 in length.

```
spacings (
  ( minArea 0.015)
  ( minAreaEdgeLength "Metal1"
    'exceptStep (0.12 0.05)
    0.017
  )
) ;spacings
```

■ Metal1



a) PASS. The minAreaEdgeLength constraint does not apply because the edge with length 0.15 (>0.12) has an adjacent edge with length 0.04 (<0.05). The minArea constraint applies and is satisfied.



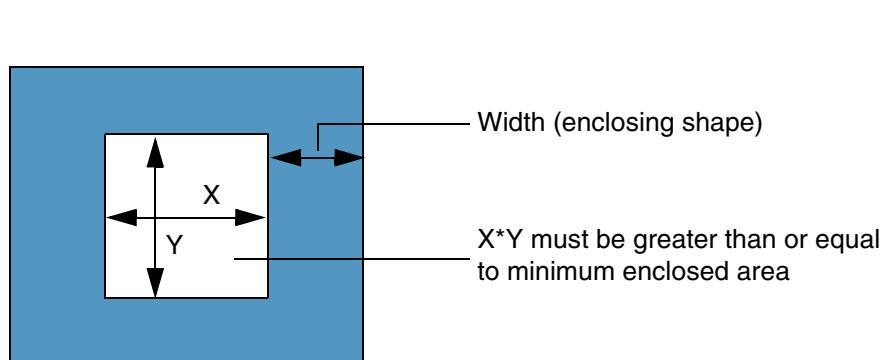
b) FAIL. The edges adjacent to the edge with length 0.15 are both 0.05 long (and not less than 0.05), and the area of the polygon is less than 0.017.

minHoleArea

```
spacings(  
    ( minHoleArea tx_layer  
      f_area  
    )  
) ;spacings  
  
spacingTables(  
    ( minHoleArea tx_layer  
      (( "width" nil nil )  
        [f_default]  
      )  
      (g_table)  
    )  
) ;spacingTables
```

Specifies the minimum area for a hole (empty area enclosed by a metal donut shape) on a layer. Enclosed areas are often associated with slotted metals or power grids.

This constraint is used in conjunction with the [minHoleWidth](#) constraint for slotting because it is possible for geometries to meet the minimum area requirement, but not the minimum spacing requirement.



Virtuoso Technology Data Constraints Reference

Area Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_area</i>	The area of the hole must be greater than or equal to this value.
"width" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of the <i>table</i> row is as follows: (<i>f_width</i> <i>f_value</i>) where, ■ <i>f_width</i> is the width of the donut shape. ■ <i>f_value</i> is the minimum area of the hole when the width of the donut shape is greater than or equal to the corresponding index. Type: A 1-D table specifying floating point width values and area values in area units

Parameters

<i>f_default</i>	The area value to be used when no table entry applies. Type: Float value with area units

Example

The area of a hole on Metal1 must be greater than or equal to 1.5 and the width of the hole in both X and Y directions must be greater than or equal to 1.0.

```
spacings (
  ( minHoleArea "Metal1"
    1.5
  )
  ( minHoleWidth "Metal1"
    1.0
  )
) ;spacings
```

minIsolatedArea

```
spacings(  
    ( minIsolatedArea tx_layer  
      ['exceptSpacing g_exceptSpacing']  
      f_area  
    )  
); spacings
```

Specifies the minimum area required for shapes when no other shape is in a defined region.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_area</i>	The minimum area in square microns.

Parameters

<i>'exceptSpacing g_exceptSpacing</i>	The constraint does not apply if another shape on the same layer is at a distance that is less than this value.

Example

The minimum area for shapes on Metal1 is 75 except when another shape on the same layer is at a distance of less than 10.

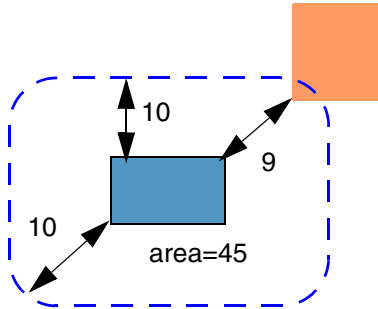
Virtuoso Technology Data Constraints Reference

Area Constraints

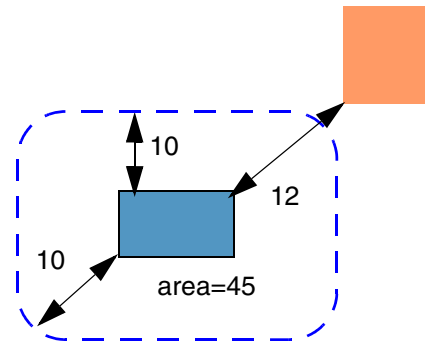
```
spacings (
  ( minIsolatedArea "Metall"
    'exceptSpacing 10
    75.0
  )
) ;spacings
```

Shape 1

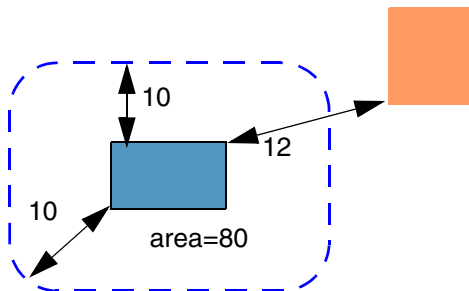
Shape 2



a) The constraint is not applicable because another shape is present at a distance of less than 10.



b) FAIL. The constraint is applicable because although no shape is present at a distance of less than 10, the area is less than 75.



PASS. The constraint is applicable because no shape is present at a distance of less than 10 and the area is more than 75.

minRectArea

```

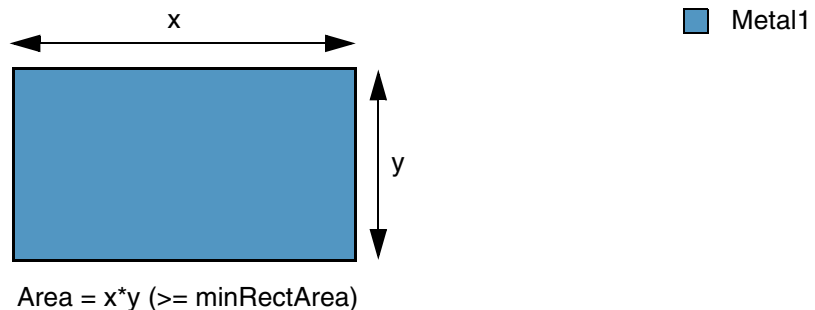
spacings (
    ( minRectArea tx_layer
      ['mask1 | 'mask2 | 'mask3]
      ['maxWidth f_width]
      ['layer tx_overlapLayer 'overlapType x_overlapType]
      ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      f_area
    )
) ;spacings

```

Specifies the minimum area allowed for a rectangular shape on the specified layer.

This constraint applies only to rectangular shapes. To specify the minimum area for non-rectangular shapes, use the [minArea](#) constraint.

Note: If both `minRectArea` and [minSize](#) are set on a layer, rectangular shapes on that layer must satisfy at least one constraint.



Virtuoso Technology Data Constraints Reference

Area Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_area</i>	The minimum area that a rectangular shape on the layer must have. Type: Float value with area units

Parameters

<code>'mask1 'mask2 'mask3</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only to the shapes on the specified mask. Type: Boolean
<code>'maxWidth f_width</code>	(Virtuoso Advanced Node for Layout Standard) The constraint applies only to shapes with width (shorter edge) less than or equal to this value.
<code>'layer tx_overlapLayer</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies to a shape only if one or both its ends overlap shapes on this layer. The number of overlaps is specified with <code>'overlapType</code> . Type: String (layer name) or Integer (layer number)
<code>'overlapType x_overlapType</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the number of overlaps is equal to this value. Valid values are 1 and 2.

Virtuoso Technology Data Constraints Reference

Area Constraints

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

(Virtuoso Advanced Node for Layout Only) Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Only) The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Only) The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Area Constraints

Examples

- Example 1: minRectArea
- Example 2: minRectArea with maxWidth
- Example 3: minRectArea with layer and overlapType

Example 1: minRectArea

The minimum area for a rectangular shape on `metal1` is 5.0, and, on `metal2`, it is equal to the value of the technology parameter `areavall`.

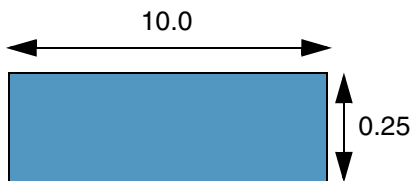
```
spacings (
  ( minRectArea "metal1" 5.0 )
  ( minRectArea "metal2" techParam("areavall") )
) ;spacings
```

Example 2: minRectArea with maxWidth

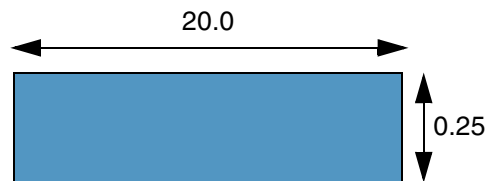
The minimum area for a rectangular shape with width less than or equal to 0.3 is 5.0.

```
spacings (
  ( minRectArea "Metal1"
    'maxWidth 0.3
    5.0
  )
) ;spacings
```

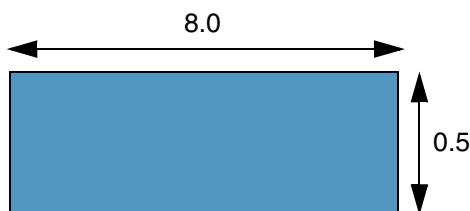
 Metal1



a) FAIL. Width is 0.25 (<0.3), but the area (10.0x0.25) is less than the minimum specified value.



b) PASS. Width is 0.25 (<0.3) and the area (20.0x0.25) is equal to the minimum specified value.



c) The constraint does not apply because the width is greater than the specified value (≤ 0.3).

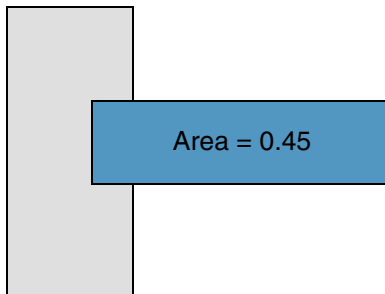
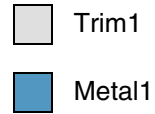
Virtuoso Technology Data Constraints Reference

Area Constraints

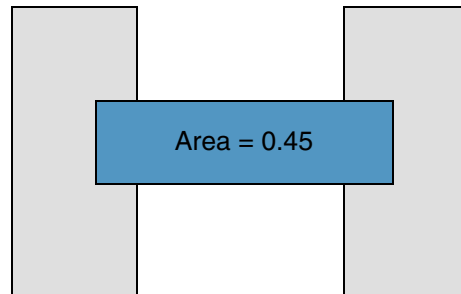
Example 3: minRectArea with layer and overlapType

The area of a Metal1 wire must be at least 0.5 if one of its ends overlaps a shape on layer Trim1 and at least 0.4 if both its ends overlap shapes on layer Trim1.

```
spacings (  
  ( minRectArea "Metal1"  
    'layer "Trim1"  
    'overlapType 1  
    0.5  
  )  
  ( minRectArea "Metal1"  
    'layer "Trim1"  
    'overlapType 2  
    0.4  
  )  
) ;spacings
```



a) FAIL. Only one end of the Metal1 shape overlaps a shape on layer Trim1, but the area of the rectangle is only 0.45 (<0.5).



b) PASS. Both ends of the Metal1 shape overlap shapes on layer Trim1 and the area of the rectangle is 0.45 (>0.4).

Density Constraints

This topic includes the following constraints:

- [maxDensity](#)
- [maxDensityCheck](#)
- [maxDiffDensity](#)
- [maxRelativeColorDensity](#)
- [minDensity](#)
- [minDensityCheck](#)
- [minRelativeColorDensity](#)

maxDensity

```

spacings (
    ( maxDensity tx_layer
      f_density
    )
) ;spacings

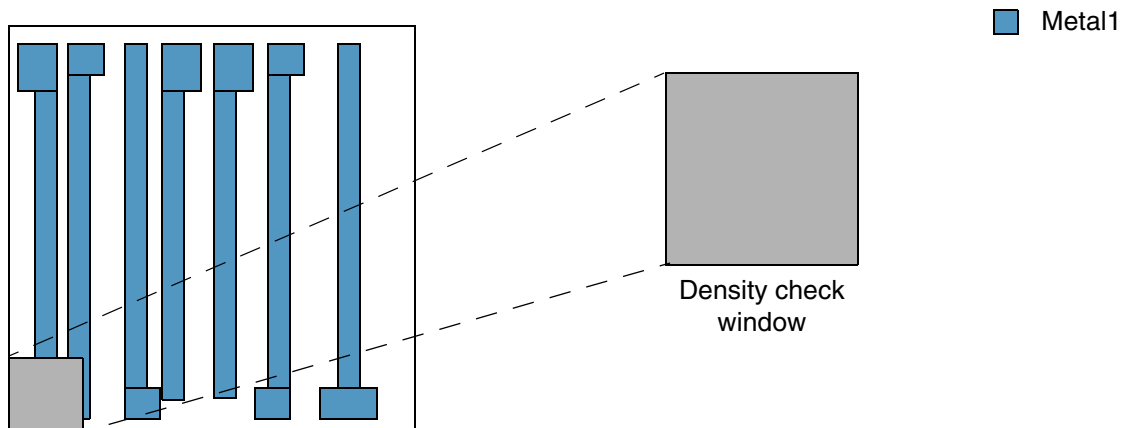
spacingTables (
    ( maxDensity tx_layer
      (( "step" nil nil ["window" nil nil] )
        [f_default]
      )
      (g_table)
    )
) ;spacingTables

```

Specifies the maximum percentage of the design area that the metal on the specified layer can occupy. Density is calculated as follows:

$$\text{Density} = \text{Metal area} / \text{Total area}$$

Density constraints are usually represented in two ways. They can apply either across the area of the whole design or specifically to the region covered by the checking window, which is moved incrementally across the design to check for localized problems.



Checking density by using a stepped value requires that you start at a corner of the design, as shown above, and validate the density percentage for an area (window) whose height and width are each twice the step size or equal to the window size. You then continue to step across the design incrementally, moving the window by the step size, each time validating that the density percentage in the window is within the maximum required value.

Virtuoso Technology Data Constraints Reference

Density Constraints

The `maxDensity` constraint is used in conjunction with the `minDensity` constraint to ensure that the metal density for a design is neither too dense nor too sparse.

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>f_density</code>	The maximum percentage of the total design area that the metal on the specified layer must occupy.
<code>"step" nil nil ["window" nil nil]</code>	This identifies the index for <i>table</i>. If the <code>window</code> table index is not specified, the density check window is a square whose sides are twice the step size. As a result, the area to be checked is equal to $(2 \times step)^2$. If the <code>window</code> table index is specified, the density check window is a square whose sides are equal to the specified window value. As a result, the area to be checked is equal to $(window)^2$.
<code>g_table</code>	The table row is defined as follows: <code>(f_step f_window) f_density</code> or <code>f_step f_density</code> where, ■ <code>f_step</code> is the step size. ■ <code>f_window</code> is the window size. ■ <code>f_density</code> is the density percentage. Type: A 1-D table specifying floating-point step and density values, or a 2-D table specifying floating-point step, window, and density values

Parameters

f_default The density value to be used when no table entry applies.

Examples

- Example 1: maxDensity with step
- Example 2: maxDensity with step and window

Example 1: maxDensity with step

The density of Metal1 as a percentage of the total area of the design must be less than or equal to 65.0 when the step size is 60.0. The density check window is a square with side equal to 120.0.

```
spacingTables(  
  ( maxDensity "Metal1"  
    (( "step" nil nil )  
      (60.0 65.0)  
    )  
)  
);spacingTables
```

Example 2: maxDensity with step and window

The density of Metal2 as a percentage of the total area of the design must be less than or equal to:

- 65.0 when the step size is 60.0 and the window size is 100.0. The density check window is a square with side equal to 100.0.
- 130.0 when the step size is 120.0 and the window size is 200.0. The density check window is a square with side equal to 200.0.

```
spacingTables(  
  ( maxDensity "Metal2"  
    (( "step" nil nil "window" nil nil )  
      65.0  
    )  
    (  
      (60.0 100.0) 65.0  
      (120.0 200.0) 130.0  
    )  
  )  
);spacingTables
```


maxDensityCheck

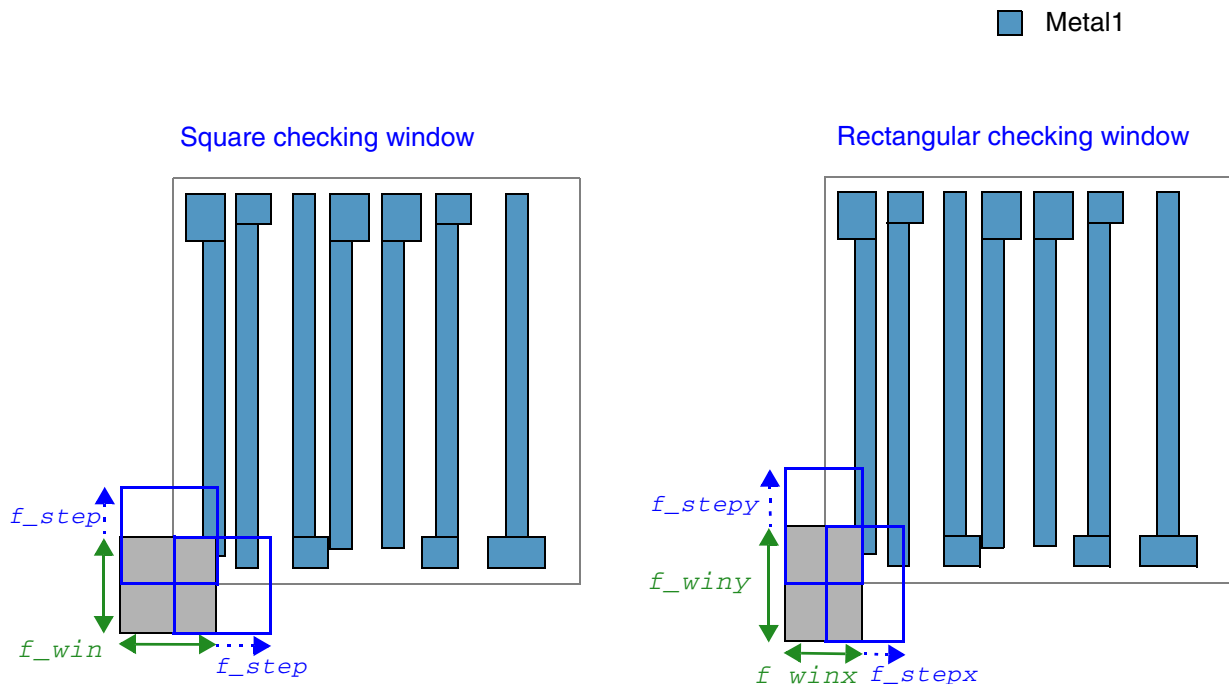
```
spacings (
    ( maxDensityCheck tx_layer
      ['step f_step | 'stepx f_stepx 'stepy f_stepy]
      ['win f_win | 'winx f_winx 'winy f_winy]
      ['mask1 | 'mask2 | 'mask3]
      f_maxDensity
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the absolute maximum density of shapes on a layer against a specified checking window that is moved by defined steps until the entire design is checked.

The absolute density of a layer is calculated as follows:

Total area of shapes in checking window / Area of checking window

Density constraints are usually represented in two ways. They can apply either across the area of the whole design or specifically to the region covered by the checking window, which is moved incrementally across the design to check for localized problems.



Checking density by using a stepped value requires that you start at a corner of the design, as shown above. The checking window may be rectangular or square. You then continue to

Virtuoso Technology Data Constraints Reference

Density Constraints

step across the design incrementally, moving the window by the step size, each time validating that the density percentage in the window is within the maximum required value.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_maxDensity</i>	The maximum density or percentage of the design area that the total shapes of a specified layer and mask must occupy.

Parameters

`'step f_step | 'stepx f_stepx 'stepy f_stepy`

The size of the steps by which the checking window is moved.

- For a square checking window, specify *f_step*, which indicates the size of the step in both x and y directions. Also, ensure that *f_win* is specified.
- For a rectangular window, specify *f_stepx* and *f_stepy*, which indicate the size of the step in x and y directions, respectively. Also, ensure that *f_winx* and *f_winy* are specified.

`'win f_win | 'winx f_winx 'winy f_winy`

The dimensions of the checking window.

- For a square checking window, specify *f_win*, which indicates the dimension of the window on each side. Also, ensure that *f_step* is specified.
- For a rectangular window, specify *f_winx* and *f_winy*, which indicate the x and y dimensions of the window, respectively. Also, ensure that *f_stepx* and *f_stepy* are specified.

`'mask1 | 'mask2 | 'mask3`

The mask to which the constraint applies.

Virtuoso Technology Data Constraints Reference

Density Constraints

Example

The maximum density of mask2 shapes on Metal1 as a percentage of the total area of the 4.62 by 7.2 window must be less than or equal to 65.2. The checking window is stepped across the entire layout in increments of half the checking window size until the entire layout has been analyzed.

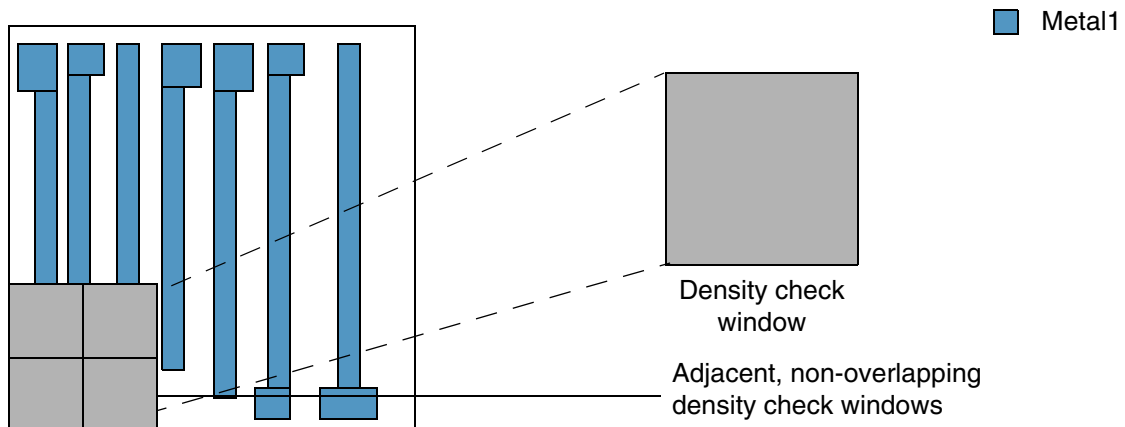
```
spacings (
  ( maxDensityCheck "Metal1"
    'winx 4.62 'winy 7.2
    'stepx 2.31 'stepy 3.6
    'mask2
    65
  )
) ;spacings
```

maxDiffDensity

```
spacingTables(  
  ( maxDiffDensity tx_layer  
    (( "step" nil nil ["window" nil nil] )  
      [f_default]  
    )  
    (g_table)  
  )  
) ;spacingTables
```

Specifies that the maximum difference in metal density in adjacent, non-overlapping density checking windows for some processes must be within a certain percentage range.

The maximum density difference between different windows is represented as a value between 0 and 100. For example, if the metal density in one window is 80% and the maximum density difference specified for adjacent density windows is 10%, then the adjacent windows can have metal density in the range 70%-90%.



Virtuoso Technology Data Constraints Reference

Density Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number) "step" nil nil ["window" nil nil] This identifies the index for <i>table</i>. If the <i>window</i> table index is not specified, the density check window is a square whose sides are twice the step size. As a result, the area to be checked is equal to $(2 \times \textit{step})^2$. If the <i>window</i> table index is specified, the density check window is a square whose sides are equal to the specified window value. As a result, the area to be checked is equal to $(\textit{window})^2$.
<i>g_table</i>	The table row is defined as follows: $(f\_step \ f\_window) \ f\_density$ or $f\_step \ f\_density$ where, ■ f_step is the step size. ■ f_window is the window size. ■ $f\_density$ is the density difference percentage. Type: A 1-D table specifying floating-point step and density values, or a 2-D table specifying floating-point step, window, and density difference values

Parameters

<i>f_default</i>	The density difference value to be used when no table entry applies.

Examples

- Example 1: maxDiffDensity with step
- Example 2: maxDiffDensity with step and window

Example 1: maxDiffDensity with step

The difference in Metal1 density in adjacent, non-overlapping density checking windows must be less than or equal to 10% when the step size is 60.0. The density check window is a square with side equal to 120.0.

```
spacingTables(  
  ( maxDiffDensity "Metal1"  
    (( "step" nil nil ))  
    (60.0 10.0)  
  )  
) ;spacingTables
```

Example 2: maxDiffDensity with step and window

The difference in Metal2 density in adjacent, non-overlapping density checking windows must be less than or equal to:

- 5% when the step size is 60.0 and the window size is 100.0. The density check window is a square with side equal to 100.0.
- 10% when the step size is 120.0 and the window size is 200.0. The density check window is a square with side equal to 200.0.

```
spacingTables(  
  ( maxDiffDensity "Metal2"  
    (( "step" nil nil "window" nil nil ))  
    (  
      (60.0 100.0) 5.0  
      (120.0 200.0) 10.0  
    )  
  )  
) ;spacingTables
```

maxRelativeColorDensity

```

spacings (
    ( maxRelativeColorDensity tx_layer
      ['step f_step | 'stepx f_stepx 'stepy f_stepy]
      ['win f_win | 'winx f_winx 'winy f_winy]
      ['mask1 | 'mask2 | 'mask3]
      f_maxDensity
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Specifies the maximum relative color density of shapes on MPT layers against a specified checking window that is moved by defined steps until the entire design is checked.

The relative color density of a layer is calculated as follows:

Total area of shapes in the set mask in the checking window / Total area of shapes in the checking window

For example, if all Metal1 shapes in a checking window are mask1, then the relative density of Metal1:mask1 is 100%.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_maxDensity</i>	The maximum relative color density or the percentage of the total design area that the metal on the specified layer must occupy.

Parameters

`'step f_step | 'stepx f_stepx 'stepy f_stepy`

The size of the steps by which the checking window is moved.

- For a square checking window, specify *f_step*, which indicates the size of the step in both x and y directions. Also, ensure that *f_win* is specified.
- For a rectangular window, specify *f_stepx* and *f_stepy*, which indicate the size of the step in x and y directions, respectively. Also, ensure that *f_winx* and *f_winy* are specified.

`'win f_win | 'winx f_winx 'winy f_winy`

The dimensions of the checking window.

- For a square checking window, specify *f_win*, which indicates the dimension of the window on each side. Also, ensure that *f_step* is specified.
- For a rectangular window, specify *f_winx* and *f_winy*, which indicate the x and y dimensions of the window, respectively. Also, ensure that *f_stepx* and *f_stepy* are specified.

`'mask1 | 'mask2 | 'mask3`

The mask to which the constraint applies.

Example

For a 4.62 by 7.2 checking window, the percentage of Metal1 shapes on mask2 should not exceed 60.

```
spacings(  
  ( maxRelativeColorDensity "Metal1"  
    'winx 4.62 'winy 7.2  
    'stepx 2.31 'stepy 3.6  
    'mask2  
    60  
  )  
) ;spacings
```

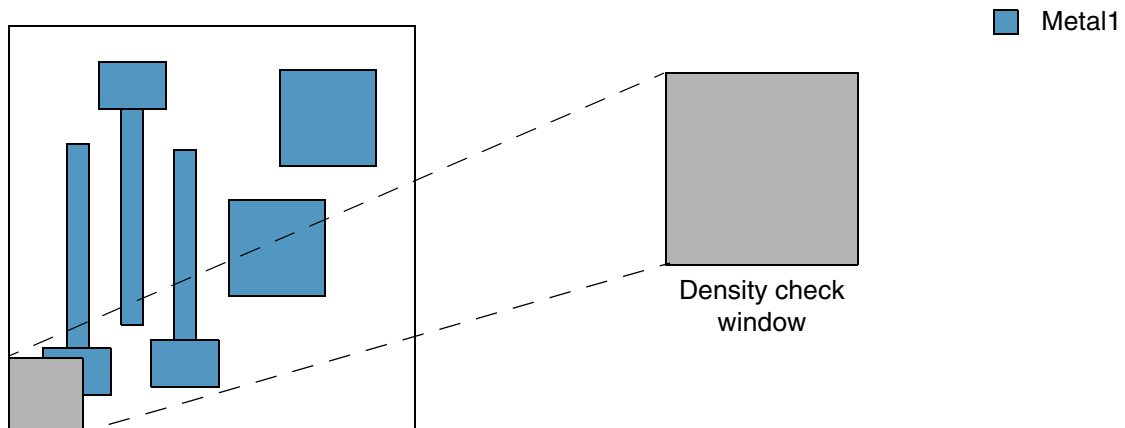

minDensity

```
spacings(  
    ( minDensity tx_layer  
      f_density  
    )  
) ;spacings  
  
spacingTables(  
    ( minDensity tx_layer  
      (( "step" nil nil ["window" nil nil] )  
        [f_default]  
      )  
      (g_table)  
    )  
) ;spacingTables
```

Specifies the minimum percentage of the design area that the metal on the specified layer must occupy. Density is calculated as follows:

$Density = Metal\ area / Total\ area$

Density constraints are usually represented in two ways. They can apply either across the area of the whole design or specifically to the region covered by the checking window, which is moved incrementally across the design to check for localized problems.



Checking density by using a stepped value requires that you start at a corner of the design, as shown above, and validate the density percentage for an area (window) whose height and width are each twice the step size or equal to the window size. You then continue to step across the design incrementally, moving the window by the step size, each time validating that the density percentage in the window exceeds the minimum required value.

Virtuoso Technology Data Constraints Reference

Density Constraints

The `minDensity` constraint is used in conjunction with the `maxDensity` constraint to ensure that the metal density for a design is neither too sparse nor too dense.

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>f_density</code>	The minimum percentage of the total design area that the metal on the specified layer must occupy.
<code>"step" nil nil ["window" nil nil]</code>	This identifies the index for <i>table</i>. If the <code>window</code> table index is not specified, the density check window is a square whose sides are twice the step size. As a result, the area to be checked is equal to $(2 \times \text{step})^2$. If the <code>window</code> table index is specified, the density check window is a square whose sides are equal to the specified window value. As a result, the area to be checked is equal to $(\text{window})^2$.
<code>g_table</code>	The table row is defined as follows: <code>(f_step f_window) f_density</code> or <code>f_step f_density</code> where, ■ <code>f_step</code> is the step size. ■ <code>f_window</code> is the window size. ■ <code>f_density</code> is the density percentage. Type: A 1-D table specifying floating-point step and density values, or a 2-D table specifying floating-point step, window, and density values

Virtuoso Technology Data Constraints Reference

Density Constraints

Parameters

f_default The density value to be used when no table entry applies.

Examples

- [Example 1: minDensity with step](#)
- [Example 2: minDensity with step and window](#)

Example 1: minDensity with step

The minimum density of Metal1 as a percentage of the total area of the design must be at least 20.0 when step size is 60.0. The density check window is a square with side equal to 120.0.

```
spacingTables(  
  ( minDensity "Metal1"  
    (( "step" nil nil )  
      20.0  
    )  
    (60.0 20.0)  
  )  
) ;spacingTables
```

Example 2: minDensity with step and window

The minimum density of Metal2 as a percentage of the total area of the design must be as follows:

- At least 20.0 when the step size is 60.0 and the window size is 100.0. The density check window is a square with side equal to 100.0.
- At least 40.0 when the step size is 120.0 and the window size is 200.0. The density check window is a square with side equal to 200.0.

```
spacingTables(  
  ( minDensity "Metal2"  
    (( "step" nil nil "window" nil nil )  
      20.0  
    )  
    (  
      (60.0 100.0) 20.0  
      (120.0 200.0) 40.0  
    )  
  )  
) ;spacingTables
```

minDensityCheck

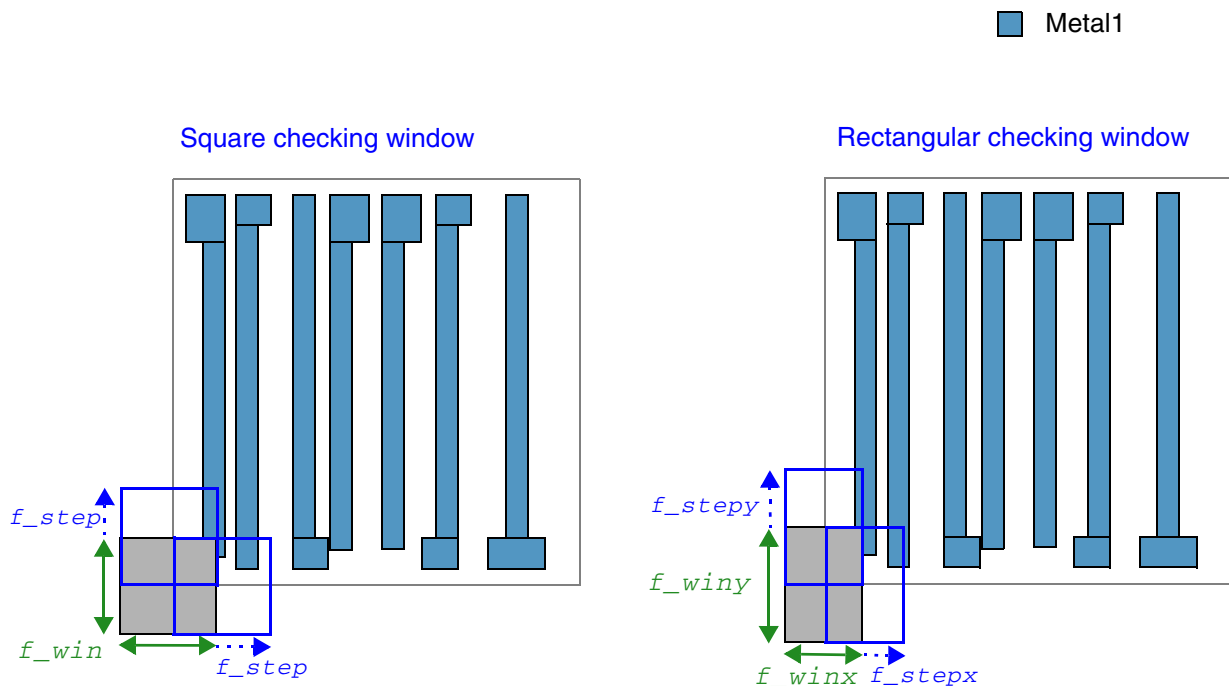
```
spacings (
    ( minDensityCheck tx_layer
      ['step f_step | 'stepx f_stepx 'stepy f_stepy]
      ['win f_win | 'winx f_winx 'winy f_winy]
      ['mask1 | 'mask2 | 'mask3]
      ['target f_target]
      f_minDensity
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the absolute minimum density of shapes on a layer against a specified checking window that is moved by defined steps until the entire design is checked.

The absolute density of a layer is calculated as follows:

Total area of shapes in checking window / Area of checking window

Density constraints are usually represented in two ways. They can apply either across the area of the whole design or specifically to the region covered by the checking window, which is moved incrementally across the design to check for localized problems.



Checking density by using a stepped value requires that you start at a corner of the design, as shown above. The checking window may be rectangular or square. You then continue to

Virtuoso Technology Data Constraints Reference

Density Constraints

step across the design incrementally, moving the window by the step size, each time validating that the density percentage in the window is within the maximum required value.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_minDensity</i>	The minimum density or percentage of the design area that the total shapes of a specified layer and mask must occupy.

Parameters

`'step f_step | 'stepx f_stepx 'stepy f_stepy`

The size of the steps by which the checking window is moved.

- For a square checking window, specify *f_step*, which indicates the size of the step in both x and y directions. Also, ensure that *f_win* is specified.
- For a rectangular window, specify *f_stepx* and *f_stepy*, which indicate the size of the step in x and y directions, respectively. Also, ensure that *f_winx* and *f_winy* are specified.

`'win f_win | 'winx f_winx 'winy f_winy`

The dimensions of the checking window.

- For a square checking window, specify *f_win*, which indicates the dimension of the window on each side. Also, ensure that *f_step* is specified.
- For a rectangular window, specify *f_winx* and *f_winy*, which indicate the x and y dimensions of the window, respectively. Also, ensure that *f_stepx* and *f_stepy* are specified.

`'mask1 | 'mask2 | 'mask3`

The mask to which the constraint applies.

Virtuoso Technology Data Constraints Reference

Density Constraints

`'target f_target`

The preferred minimum density, which should be more than or equal to `f_minDensity`. Checker treats this value as the minimum density. It acts as a buffer and ensures that all windows are slightly above the required minimum density.

Example

The minimum density of mask2 shapes on Metal1 as a percentage of the total area of a 4.62 by 7.2 checking window must be greater than or equal to 25.2%. The checking window is stepped across the entire layout in increments of half the checking window size until the entire layout has been analyzed. Because a target density is indicated, the checker reports any window that has density less than the target 32%.

```
spacings (
  ( minDensityCheck "Metal1"
    'winx 4.62 'winy 7.2
    'stepx 2.31 'stepy 3.6
    'mask2
    'target 32
    25.2
  )
) ;spacings
```

minRelativeColorDensity

```
spacings (
    ( minRelativeColorDensity tx_layer
      ['step f_step | 'stepx f_stepx 'stepy f_stepy]
      ['win f_win | 'winx f_winx 'winy f_winy]
      ['mask1 | 'mask2 | 'mask3]
      ['target f_target]
      f_minDensity
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the minimum relative color density of shapes on MPT layers against a specified checking window that is moved by defined steps until the entire design is checked.

The relative color density of a layer is calculated as follows:

Total area of shapes in the set mask in the checking window / Total area of shapes in the checking window

For example, if all Metal1 shapes in a checking window are mask1, then the relative density of Metal1:mask1 is 100%.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_minDensity</i>	The minimum relative color density or the percentage of the total design area that the metal on the specified layer must occupy.

Parameters

`'step f_step | 'stepx f_stepx 'stepy f_stepy`

The size of the steps by which the checking window is moved.

- For a square checking window, specify *f_step*, which indicates the size of the step in both x and y directions. Also, ensure that *f_win* is specified.
- For a rectangular window, specify *f_stepx* and *f_stepy*, which indicate the size of the step in x and y directions, respectively. Also, ensure that *f_winx* and *f_winy* are specified.

`'win f_win | 'winx f_winx 'winy f_winy`

The dimensions of the checking window.

- For a square checking window, specify *f_win*, which indicates the dimension of the window on each side. Also, ensure that *f_step* is specified.
- For a rectangular window, specify *f_winx* and *f_winy*, which indicate the x and y dimensions of the window, respectively. Also, ensure that *f_stepx* and *f_stepy* are specified.

`'mask1 | 'mask2 | 'mask3`

The mask to which the constraint applies.

`'target f_target`

The preferred minimum density, which should be more than or equal to *f_minDensity*. Checker treats this value as the minimum density. It acts as a buffer and ensures that all windows are slightly above the required minimum density.

Example

For a 4.62 by 7.2 checking window, Metal1 shapes on mask2 should be at least 40%.

```
spacings(  
  ( minRelativeColorDensity "Metal1"  
    'winx 4.62 'winy 7.2  
    'stepx 2.31 'stepy 3.6  
    'mask2  
    60  
  )  
)
```


Virtuoso Technology Data Constraints Reference

Density Constraints

Virtuoso Technology Data Constraints Reference

Density Constraints

Extension Constraints

This topic includes the following constraints:

- [duyExmmntension](#)
- [lineEndAlignedWithCut](#)
- [maxExtension](#)
- [minCenterlineExtension](#)
- [minCornerExtension](#)
- [minEndOfLineAdjExtension](#)
- [minEndOfLineEdgeExtension](#)
- [minEndOfLineExtension](#)
- [minExtensionDistance](#)
- [minExtensionEdge](#)
- [minExtensionEolSpacing](#)
- [minExtensionOnLongSide](#)
- [minExtensionToCenterLine](#)
- [minExtensionToCorner](#)
- [minGateExtension](#)
- [minInnerVertexProximityExtension](#)
- [minInsideCornerExtension](#)
- [minNeighborExtension](#)
- [minOppEndOfLineExtension](#)
- [minOppExtension](#)

Virtuoso Technology Data Constraints Reference

Extension Constraints

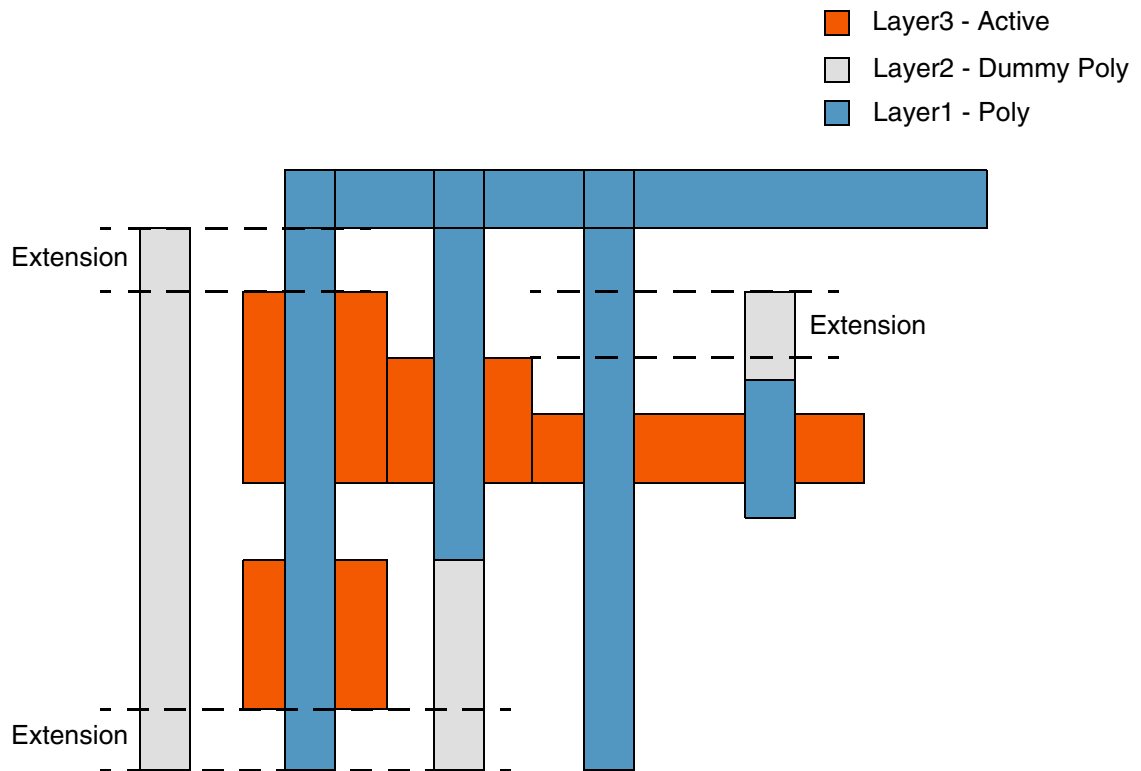
- minPRBoundaryExtension
- minProtrusionWidthWithVia
- minQuadrupleExtension
- minSideExtension
- minTouchingDirEnclosure
- minViaExtension
- minViaJointExtension
- minVoltageExtension
- minWireExtension

duyExmmtension

```
orderedSpacings (
    ( dummyExtension tx_layer1 tx_layer2 tx_layer3
      ['width f_width]
      f_extension
    )
) ;orderedSpacings
```

Specifies the minimum extension of a dummy poly shape (*layer2*) past an adjacent shape on the active layer (*layer3*). The first layer is the poly layer, the second is the dummy poly layer, and the third is the active layer.

The constraint does not apply if a gate is joined to another gate above or below.



Virtuoso Technology Data Constraints Reference

Extension Constraints

Values

<i>tx_layer1</i>	The poly layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The dummy poly layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer3</i>	The active layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The dummy poly extension must be greater than or equal to this value.

Parameters

<i>f_width</i>	The constraint applies only if the width of the dummy poly shape is less than this value.

Example

The minimum extension of a PolyX shape past a Diff shape must be at least 1.0 if the PolyX shape has width less than 0.5.

```
orderedSpacings(  
  ( dummyExtension "Poly1" "PolyX" "Diff"  
    'width 0.5  
    1.0  
  )  
) ;orderedSpacings
```

Virtuoso Technology Data Constraints Reference

Extension Constraints

finExtension

```
spacings (
  ( finExtension tx_layer
    'direction
    ['spanLengthRanges (g_spanLenRange1 g_spanLenRange2 ...
g_spanLenRangeN) ]
    ['insideLayers (tx_layer2 tx_layer3 ... tx_layerN)
    ['insidePurposes (t_purpose2 t_purpose3 ... t_purposeN) ]
    ['finIndexes (t_snapPattern1 t_snapPattern2 ... t_snapPatternN) ]
    t_snapPatternDef
  )
) ;spacings
```

Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

t_snapPatternDef

Allowed snap pattern definitions. Every entry references a snapPatternDef or widthSpacingSnapPatternDef by name.

Type: A list of snap pattern definitions, each enclosed in double quotes and separated by a space

Parameters

'direction

The edge of the shape that should be checked. Valid values: low and high.

'spanLengthRanges (g_spanLenRange1 g_spanLenRange2 ... g_spanLenRangeN)

The constraint is applied only for the specified ranges of span lengths.

'insideLayers (tx_layer2 tx_layer3 ... tx_layerN)

Virtuoso Technology Data Constraints Reference

Extension Constraints

The constraint applies only if the shapes on *tx_layer* overlap a shape on one of these layers.

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose2 t_purpose3 ... t_purposeN)`

The `insideLayers` parameter is limited to the specified purposes.

Type: String (purpose name)

`'finIndexes`

The snap pattern indexes from where the shape should start.

Example

```
(finExtension "Active"
'low
  'spanLengthRanges ("[0.4 0.7]" "1")
  'insideLayers ("matching_grid1")
  'finIndexes (1 2)
  "spDef"
)
```


lineEndAlignedWithCut

```
orderedSpacings (
    ( lineEndAlignedWithCut tx_metalLayer tx_cutLayer
      'edgeExtension f_edgeExtension
      'width f_width
      'distance f_distance
      'cutOnAnyWires
      'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
      (g_length)
    )
) ;orderedSpacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies that a line end with a cut of specified enclosure, which is aligned against two other line ends on both sides equal to the specified value would be a violation.

Values

<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_length</i>	The distance the sides of the search window extend beyond the cut edges is equal to this value.

Parameters

<i>'edgeExtension f_edgeExtension</i>	The constraint applies when the extension is less than or equal to the specified value.
<i>'width f_width</i>	The constraint applies when the width of all the three wires is equal to the specified value.
<i>'distance f_distance</i>	The constraint applies when the distance or spacing between two neighbour wires is equal to the specified value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

'cutOnAnyWires The constraint applies when the cut is on any of the three wires.

'cutClass {*f_cutWidth* | (*f_cutWidth* *f_cutLength*) | *t_name*}

The cut class to which the constraint applies, specified by width, by length, or by name (as defined in a [cutClasses](#) constraint).

- *f_cutWidth*: Width
- *f_cutLength*: Length
- *t_name*: Name of the cut class

matchingFinIndex

```

spacings (
    ( matchingFinIndex tx_layer
      'direction
      ['insideLayers (tx_layer2 tx_layer3 ... tx_layerN)
      ['insidePurposes (t_purpose2 t_purpose3 ... t_purposeN)]
      ['sameLength]
      t_snapPatternDef
    )
) ; spacings

```

```

spacings(
    ( matchingFinIndex tx_layer
      'direction
      'insideLayers (tx_layer2 tx_layer3 . . . tx_layerN)
      ['insidePurposes (t_purpose2 t_purpose3 . . . t_purposeN)]
      ['sameLength]
      t_snapPatternDefName
    )
) ; spacings

```

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t_snapPatternDef</i>	Allowed snap pattern definitions. Every entry references a snapPatternDef or widthSpacingSnapPatternDef by name. Type: A list of snap pattern definitions, each enclosed in double quotes and separated by a space

Parameters

'direction	The edge of the shape that should be checked. Valid values: low and high.
'spanLengthRanges (g_spanLenRange1 g_spanLenRange2 ... g_spanLenRangeN)	

Virtuoso Technology Data Constraints Reference

Extension Constraints

The constraint is applied only for the specified ranges of span lengths.

`'insideLayers (tx_layer2 tx_layer3 ... tx_layerN)`

The constraint applies only if the shapes on *tx_layer* overlap a shape on one of these layers.

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose2 t_purpose3 ... t_purposeN)`

The *insideLayers* parameter is limited to the specified purposes.

Type: String (purpose name)

`'finIndexes` The snap pattern indexes from where the shape should start.

Example

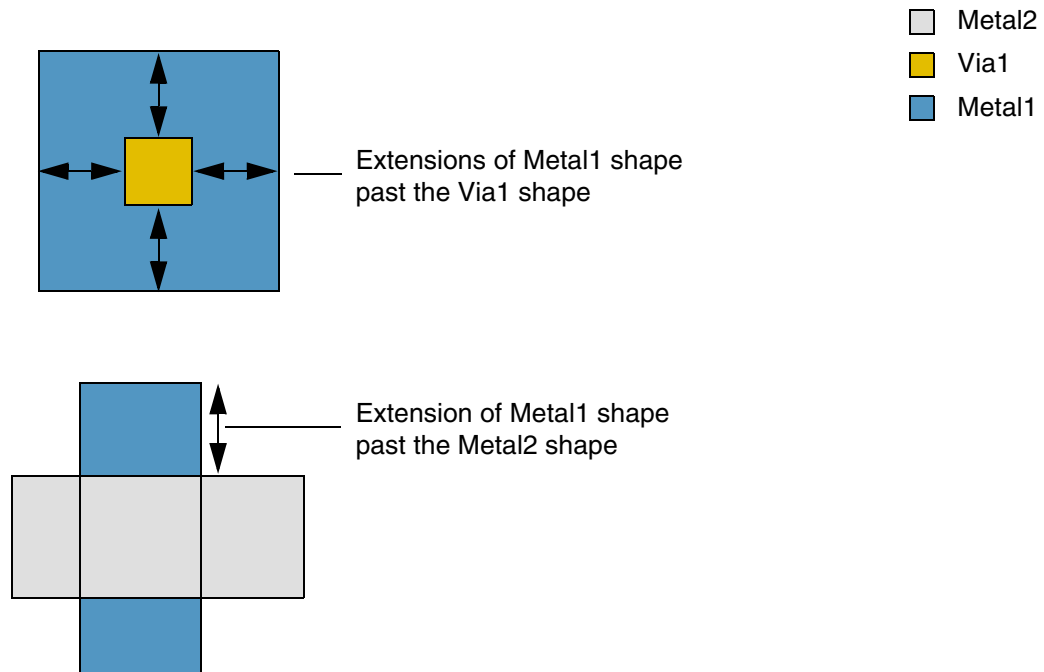
```
(finExtension "Active"
'low
'spanLengthRanges ("[0.4 0.7]" "1")
'insideLayers ("matching_grid1")
'finIndexes (1 2)
"spDef"
)
```

maxExtension

```
orderedSpacings (
    ( maxExtension tx_layer1 tx_layer2
      f_extension
    )
) ;orderedSpacings

spacingTables (
    ( maxExtension tx_layer1 tx_layer2
      (( "width" nil nil )
        [f_default]
      )
      g_table
    )
) ;spacingTables
```

Specifies the maximum extension of the shape on the first layer past the shape on the second layer.



Virtuoso Technology Data Constraints Reference

Extension Constraints

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension must be less than this value.
"width" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of the table is as follows: (<i>f_width f_extension</i>) where, ■ <i>f_width</i> is the width of the <i>layer1</i> shape. The width of the <i>layer1</i> shape must be greater than this value for the corresponding extension value to apply. ■ <i>f_extension</i> is the extension of the <i>layer1</i> shape past the <i>layer2</i> shape. The extension must be less than this value. Type: A 1-D table specifying floating-point width and extension values

Parameters

<i>f_default</i>	The extension value to be used when no table entry applies.

Examples

- Example 1: maxExtension
- Example 2: maxExtension with 1-D table

Example 1: maxExtension

The extension of a Metal1 shape past a Metal2 shape must be less than 3.0.

```
orderedSpacings(  
  ( maxExtension "Metal1" "Metal2"  
    3.0)  
)  
);orderedSpacings
```

Example 2: maxExtension with 1-D table

The extension of a Metal2 shape past a Via2 shape must be as follows:

- Less than 0.21 if the width of the Metal2 shape is greater than 0.0
- Less than 0.22 if the width of the Metal2 shape is greater than 0.1

```
spacingTables(  
  ( maxExtension "Metal2" "Via2"  
    (( "width" nil nil ))  
    (  
      0.0 0.21  
      0.1 0.22  
    )  
  )  
)  
);spacingTables
```

minCenterlineExtension

```
orderedSpacings (
  ( minCenterlineExtension tx_layer1 tx_layer2
    'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
    'maxEffectiveWidth f_maxEffectiveWidth
    ['endOfLineWidthRange (f_width1 f_width2)]
    ['distanceWithin f_distance 'exactWidth f_exactWidth]
  )
) ;orderedSpacings
```

Requires that via cuts of the specified cut class must lie on the centerline of the *layer1* and *layer2* shapes that the via cut connects if the width of the shapes is less than the specified value.

For a single-cut via, the center of the via cut must lie on the centerline of the shape with width less than *maxEffectiveWidth*. Whereas, for aligned double-cut vias, the center of the bounding box of the two aligned via cuts must lie on the centerline of the shape with width less than *maxEffectiveWidth*, provided all other specified conditions are met. If both shapes that the via cut connects satisfy the width condition, the center of the via cut or the bounding box must lie at the intersection of the centerlines of the two shapes.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

`'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'maxEffectiveWidth f_maxEffectiveWidth`

The center of the via cut must lie on the centerline of the shape with width less than or equal to this value. If both shapes have width less than or equal to this value, the center of the via cut must lie at the intersection of the centerlines of the two shapes.

`'endOfLineWidthRange (f_width1 f_width2)`

The centerline cannot intersect an end-of-line edge if its width falls in this range.

Type: Floating-point values specifying a [range](#) of widths

`'distanceWithin f_distance 'exactWidth f_exactWidth`

The center of the bounding box of aligned double-cut vias that are less than or equal to *distance* apart must lie on the centerline of the shape with width equal to *exactWidth*.

Examples

- [Example 1: minCenterlineExtension with cutClass and maxEffectiveWidth](#)
- [Example 2: minCenterlineExtension with cutClass, maxEffectiveWidth, distanceWithin, and exactWidth](#)

Virtuoso Technology Data Constraints Reference

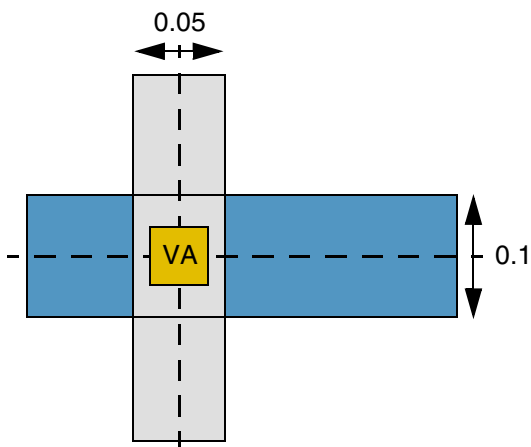
Extension Constraints

Example 1: *minCenterlineExtension* with *cutClass* and *maxEffectiveWidth*

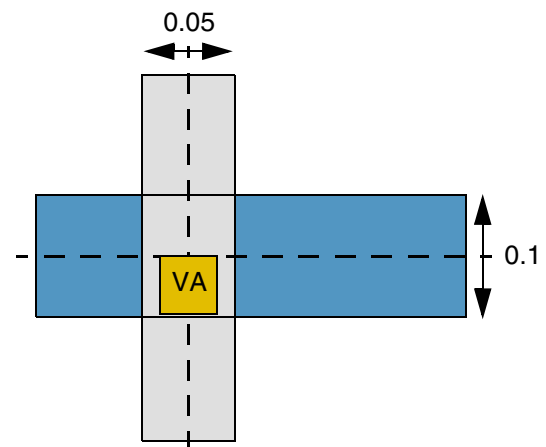
The center of a VA via cut must lie on the centerline of Metal1 and Metal2 shapes if the width of the shapes is less than or equal to 0.13.

```
orderedSpacings (
  ( minCenterlineExtension "Metal1" "Metal2"
    'cutClass "VA"
    'maxEffectiveWidth 0.13
  )
) ;orderedSpacings
```

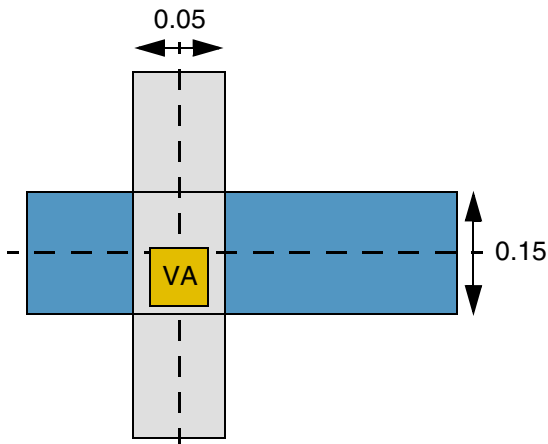
Metal2
 Via1
 Metal1



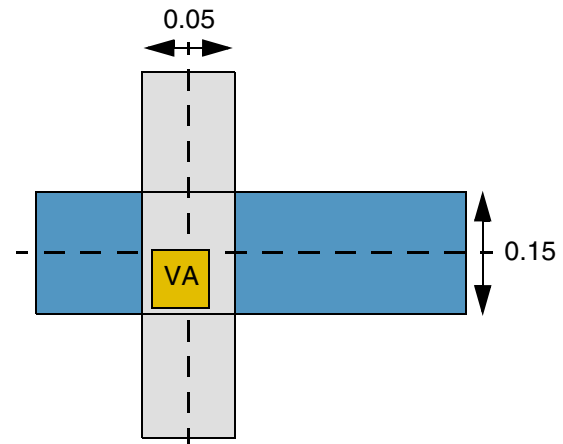
a) PASS. Both Metal1 and Metal2 shapes are less than 0.13 in width, and the center of the via cut lies on the centerlines of both shapes.



b) FAIL. Both Metal1 and Metal2 shapes are less than 0.13 in width, but the center of the via cut does not lie on the centerline of the Metal1 shape.



c) PASS. The width of the Metal2 shape is less than 0.13 and the center of the via cut lies on the centerline of this shape. The constraint does not apply to the Metal1 shape because its width is greater than 0.13.



d) FAIL. The width of the Metal2 shape is less than 0.13, but the center of the via cut does not lie on the centerline of this shape. The constraint does not apply to the Metal1 shape because its width is greater than 0.13.

Virtuoso Technology Data Constraints Reference

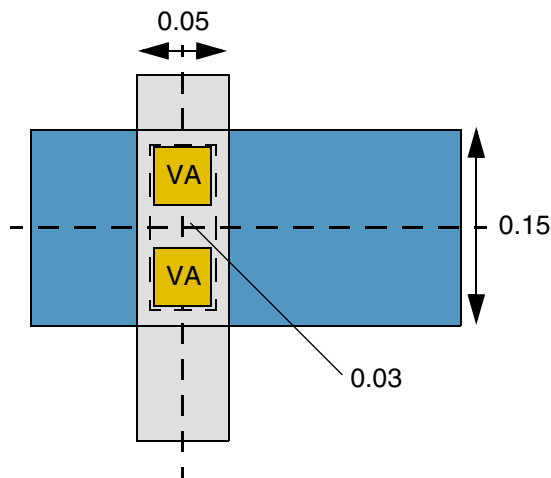
Extension Constraints

Example 2: *minCenterlineExtension* with *cutClass*, *maxEffectiveWidth*, *distanceWithin*, and *exactWidth*

The center of the bounding box of aligned double-cut VA vias that are less than or equal to 0.03 apart must lie on the centerline of the shape with width equal to 0.15.

```
orderedSpacings (  
  ( minCenterlineExtension "Metal1" "Metal2"  
    'cutClass "VA"  
    'maxEffectiveWidth 0.13  
    'distanceWithin 0.03 'exactWidth 0.15  
  )  
) ;orderedSpacings
```

□ Metal2
■ Via1
■ Metal1



PASS. The distance between the via cuts is 0.03 and the width of the Metal1 shape is equal to 0.15. The center of the bounding box of the via cuts lies on the centerline of the Metal1 shape.

minCornerExtension

```
orderedSpacings (
    ( minCornerExtension tx_metalLayer tx_cutLayer
      'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
      'corners x_corners
      f_extension
    )
) ;orderedSpacings
```

Specifies a search window whose sides extend up to a distance equal to *extension* from the cut edges. A violation occurs if the number of metal concave corners inside the search window exceeds *corners*.

Note: If a metal corner collides with a corner of the search window, it is not counted; if a metal corner aligns with a side edge of the search window, it is counted.

Values

<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The distance the sides of the search window extend beyond the cut edges is equal to this value.

Parameters

```
'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
```

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

Virtuoso Technology Data Constraints Reference

Extension Constraints

'corners x_corners

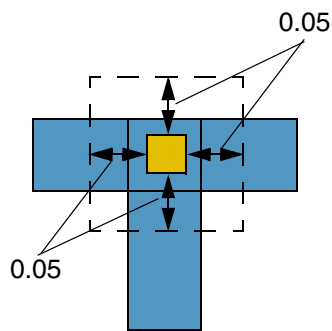
A violation occurs if the number of metal concave corners inside the search window exceeds this value.

Example

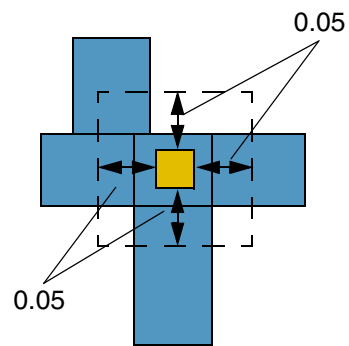
A violation occurs if more than two concave corners are found inside a search window whose sides extend 0.05 away from the edges of a VA via cut.

```
orderedSpacings (  
  ( minCornerExtension "Metal1" "Via1"  
    'cutClass "VA"  
    'corners 2  
    0.05  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1



a) PASS. Only two concave corners are present inside the dotted search window.



b) FAIL. More than two concave corners are present inside the dotted search window.

minEndOfLineAdjExtension

```
orderedSpacings (
    ( minEndOfLineAdjExtension tx_metalLayer tx_cutLayer
      'eolSpacing f_eolSpacing
      'width f_width
      ['orthogonalPair]
      f_extension
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Only) Specifies a cut extension when an adjacent shape within a specified distance is present.

Values

tx_metalLayer tx_cutLayer

The layers on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_extension

The minimum required extension value.

Parameters

'eolSpacing *f_eolSpacing*

The constraint applies when the adjacent shape is within this distance from the end-of-line.

'width *f_width*

The constraint applies to shapes with an end-of-line less than or equal to this value.

'orthogonalPair

The constraint applies to orthogonal pairs of shapes.

Virtuoso Technology Data Constraints Reference

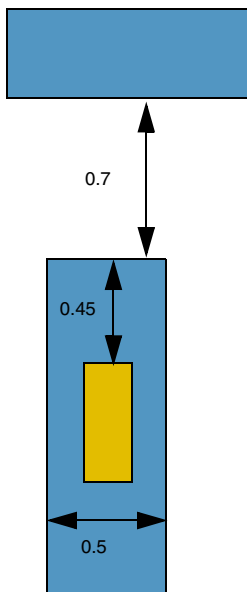
Extension Constraints

Example

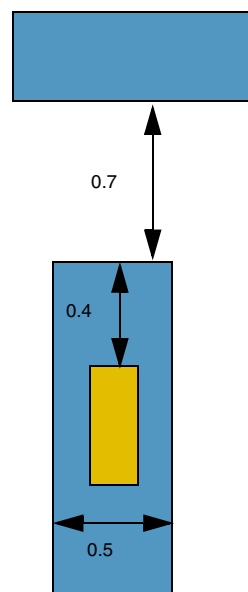
This example specifies the required extension of M1 over V1 to be 0.45 when another M1 shape is present within 0.7 of the end-of-line edge in the orthogonal routing direction when the enclosing shape has a width of less than or equal to 0.5.

```
orderedSpacings (  
  (minEndOfLineAdjExtension "M1" "V1"  
    'width 0.5  
    'eolSpacing 0.7  
    'orthogonalPair  
      0.45  
  )  
) ;orderedSpacings
```

■ M1
■ V1



a) PASS: The constraint applies because an orthogonal shape is within 0.7 and the width is 0.5. The constraint is met because the extension is greater than or equal to 0.45.



b) FAIL: The constraint also applies, but the extension is 0.4, which is less than the 0.45 minimum.

minEndOfLineEdgeExtension

```
orderedSpacings (
  ( minEndOfLineEdgeExtension tx_layer1 tx_layer2
    ['cutClass {f_width | (f_width f_length) | t_name}]
    'width f_eolWidth ['minWidth f_minWidth]
    ['horizontalEdge | 'verticalEdge]
    ['equalRectWidth]
    ['length f_length | 'exactExtension f_exactExtension | 'longEdgeOnly
      | 'shortEdgeOnly
    ]
    ['sideExclusion (f_sideEdgeExclusion f_backwardExclusion
      f_forwardExclusion)
    ]
    ['maxRectEnclosureArea f_area]
    ['allSides]
    ['includeEdgeAligned]
    ['concaveCornerLength f_concaveCornerLength]
    ['exceptSpacing f_spacing]
    ['minVoltage f_minVoltage]
    ['maxVoltage f_maxVoltage]
    f_extension
  )
) ;orderedSpacings
```

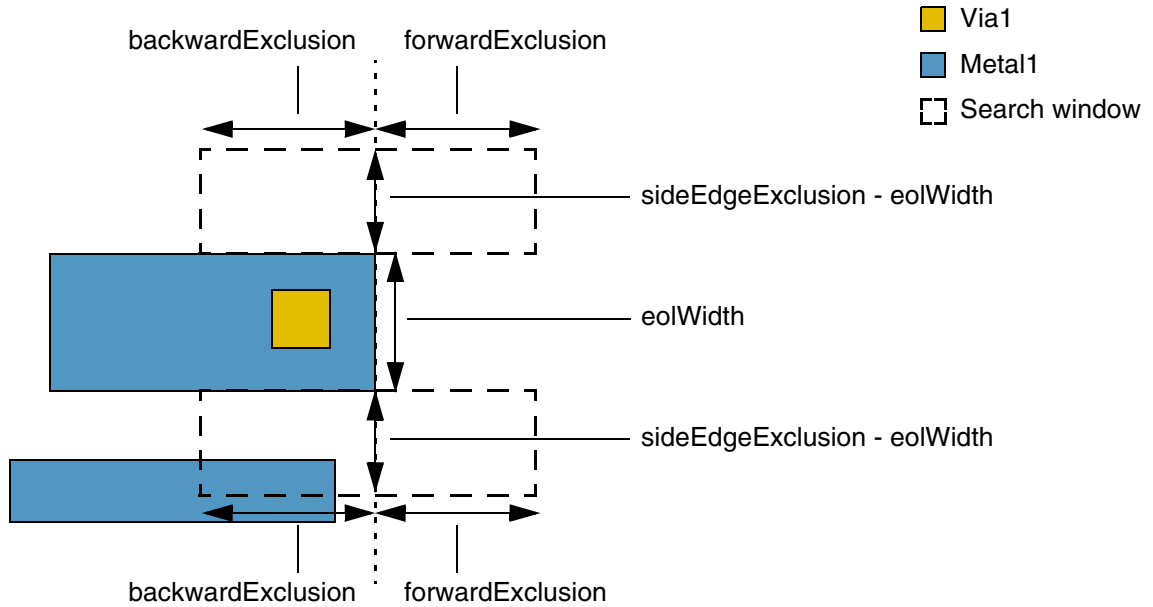
Specifies the minimum extension of an end-of-line edge of a shape on *layer2* past a shape on *layer1*. The end-of-line edge must be less than *eolWidth* wide and the shape on *layer1* must be of the specified cut class.

Optionally, an exact extension value can be specified in addition to the minimum extension value, or the constraint can be applied to either the long or short edges of a rectangular cut class. The optional *'sideExclusion* parameter defines a search window, as shown below; the constraint applies only if a neighboring parallel shape is present inside this search window.

Virtuoso Technology Data Constraints Reference

Extension Constraints

This constraint can be used with region ID layers.



Virtuoso Technology Data Constraints Reference

Extension Constraints

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The minimum required extension value.

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

`'width f_eolWidth`

The constraint applies only if the end-of-line width of the *layer2* shape is less than this value.

`'minWidth f_minWidth`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the end-of-line width of the *layer2* shape is greater than or equal to this value (and less than *eolWidth*) and the end-of-line edge touches a wire with width greater than or equal to this value.

`'horizontalEdge | 'verticalEdge`

The constraint applies only if the line-end edge is in the specified direction. If not specified, the extension is applied on all four sides of the cut.

`'equalRectWidth`

The constraint applies to side edges with length less than *eolWidth* only if the length is equal to the width of the wire.

Virtuoso Technology Data Constraints Reference

Extension Constraints

'exactExtension *f_exactExtension*

The extension must be equal to this value. Otherwise, the extension must be greater than or equal to *extension*.

'longEdgeOnly

The extension must be on the long edge of a rectangular cut.

'shortEdgeOnly

(Virtuoso Advanced Node for Layout Only) The extension must be on the short edge of a rectangular cut.

'sideExclusion (*f_sideEdgeExclusion f_backwardExclusion f_forwardExclusion*)

The constraint applies only if a neighboring shape parallel to the enclosing shape is present inside the search window, which extends both forward (*forwardExclusion*) and backward (*backwardExclusion*) and on both sides (*sideEdgeExclusion - eolWidth*) of the end-of-line edge.

'length *f_length*

The constraint applies only if the length of both sides of the enclosing shape is greater than or equal to this value.

'maxRectEnclosureArea *f_area*

The constraint applies only if the enclosing shape is rectangular and its area is less than this value.

'allSides

The extension is applied on all four sides of the cut.

'includeEdgeAligned (Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if a cut edge is aligned to end-of-line edge.

'concaveCornerLength *f_concaveCornerLength*

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.

'exceptSpacing *f_spacing*

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply when the line-end spacing of the neighbor wire is equal to the given value.

'minVoltage *f_minVoltage*

Virtuoso Technology Data Constraints Reference

Extension Constraints

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the voltage on the shape is greater than or equal to this value.

`'maxVoltage f_maxVoltage`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the voltage on the shape is less than or equal to this value.

Examples

- Example 1: minEndOfLineEdgeExtension with cutClass, width, equalRectWidth, and longEdgeOnly
- Example 2: minEndOfLineEdgeExtension with cutClass, width, equalRectWidth, and exactExtension
- Example 3: minEndOfLineEdgeExtension with cutClass, width, and maxRectEnclosureArea
- Example 4: minEndOfLineEdgeExtension with cutClass, width, and sideExclusion
- Example 5: minEndOfLineEdgeExtension with cutClass, width, and concaveCornerLength
- Example 6: minEndOfLineEdgeExtension with cutClass, width, minWidth, and shortEdgeOnly
- Example 7: minEndOfLineEdgeExtension with minVoltage and maxVoltage

Virtuoso Technology Data Constraints Reference

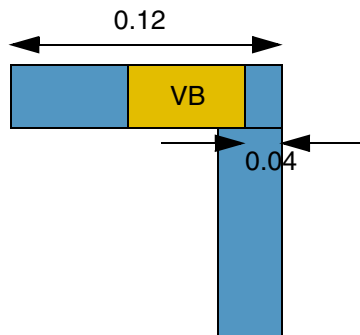
Extension Constraints

Example 1: minEndOfLineEdgeExtension with cutClass, width, equalRectWidth, and longEdgeOnly

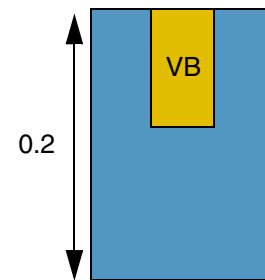
The long edge of a rectangular cut of type VB must have an extension of at least 0.04 on an end-of-line edge less than 0.15 wide.

```
orderedSpacings (  
  ( minEndOfLineEdgeExtension "Via1" "Metal1"  
    'cutClass "VB"  
    'width 0.15  
    'longEdgeOnly  
    0.04  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1



a) FAIL. The long edge of the cut is on an end-of-line edge. However, the Metal1 extension on the long edge of the cut is zero, instead of 0.04.



b) The constraint does not apply because the long edge of the cut is not on an end-of-line edge.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 2: *minEndOfLineEdgeExtension* with *cutClass*, *width*, *equalRectWidth*, and *exactExtension*

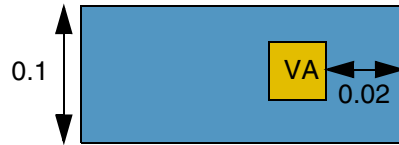
A cut of type VA must have an extension of at least 0.05 on an end-of-line edge less than 0.15 wide. An extension of 0.0 is also allowed. This extension requirement does not apply to a long edge of the enclosing Metal1 shape with length less than 0.15 if the length is not equal to the width of the wire.

```
orderedSpacings (  
  ( minEndOfLineEdgeExtension "Via1" "Metal1"  
    'cutClass "VA"  
    'width 0.15  
    'equalRectWidth  
    'exactExtension 0.0  
    0.05  
  )  
);orderedSpacings
```

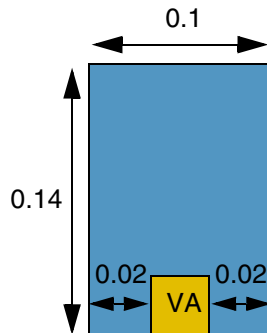
 Via1
 Metal1



a) PASS. The end-of-line width is 0.1 (< 0.15) and an exact extension value of 0.0 is allowed.



b) FAIL. The end-of-line width is 0.1 (< 0.15). However, the Metal1 extension is 0.02 (< 0.05 and not equal to 0).



c) The constraint does not apply to the vertical edges because their length is greater than the width of the wire (which makes them sides, and not end-of-line edges). The bottom edge has a zero extension, which is allowed. A violation would occur if 'equalRectWidth' was not specified.

Virtuoso Technology Data Constraints Reference

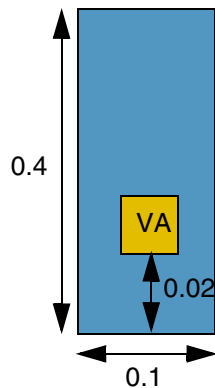
Extension Constraints

Example 3: *minEndOfLineEdgeExtension* with *cutClass*, *width*, and *maxRectEnclosureArea*

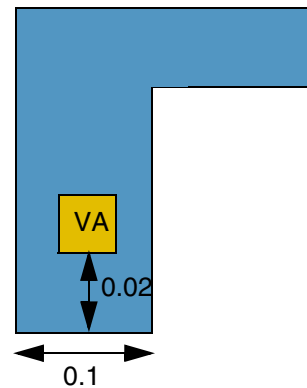
A cut of type VA must have an extension of at least 0.025 if the end-of-line edge is less than 0.2 wide and the enclosing Metal1 shape is a rectangle with area less than 0.05.

```
orderedSpacings (  
  ( minEndOfLineEdgeExtension "Via1" "Metal1"  
    'cutClass "VA"  
    'width 0.2  
    'maxRectEnclosureArea 0.05  
    0.025  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1



a) FAIL. The constraint applies because the end-of-line width is 0.1 (< 0.2) and the area of the enclosing shape is 0.04 (< 0.05). However, the Metal1 extension is 0.02 (< 0.025).



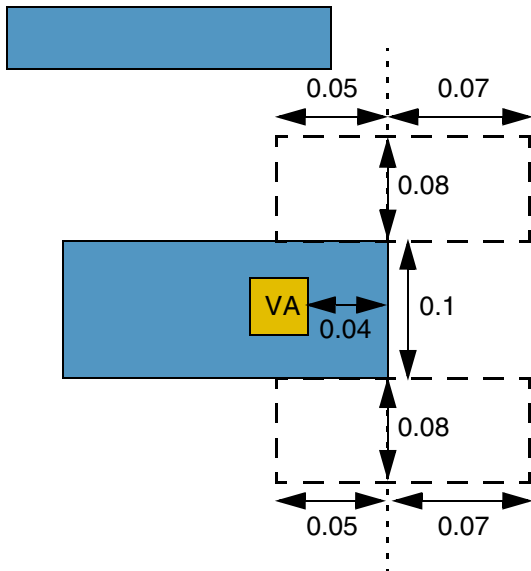
b) The constraint does not apply because the enclosing shape is not rectangular.

Example 4: minEndOfLineEdgeExtension with cutClass, width, and sideExclusion

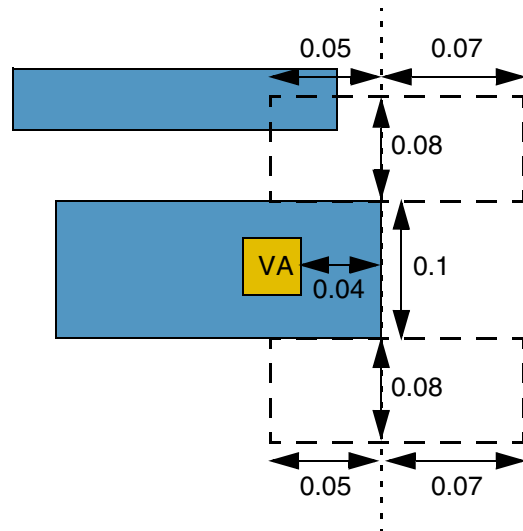
A cut of type VA must have an extension of at least 0.05 if the end-of-line edge is less than 0.15 wide and a neighboring parallel shape is present inside a search window with dimensions 0.18 (side edge exclusion), 0.05 (backward exclusion), and 0.07 (forward exclusion).

```
orderedSpacings (
  ( minEndOfLineEdgeExtension "Via1" "Metal1"
    'cutClass "VA"
    'width 0.15
    'sideExclusion (0.18 0.05 0.07)
    0.05
  )
) ;orderedSpacings
```

Via1
 Metal1
 Search window



a) The constraint does not apply because the neighboring parallel shape is outside the search window.



b) FAIL. The constraint applies because the end-of-line width is 0.1 (<0.15) and the neighboring parallel shape is present inside the search window. However, the Metal1 extension is 0.04 (<0.05).

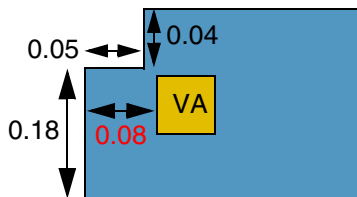
Example 5: minEndOfLineEdgeExtension with cutClass, width, and concaveCornerLength

A VA via cut on layer Via1 must have a Metal1 end-of-line edge extension greater than or equal to 0.09 if the following conditions are met:

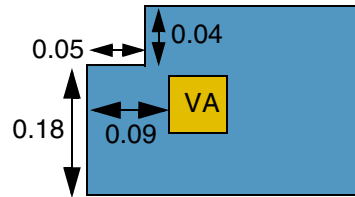
- The end-of-line edge is less than 0.2 wide.
- The edge adjoining the end-of-line edge forms a concave corner, with both concave edges less than 0.06 in length.

```
orderedSpacings (
  ( minEndOfLineEdgeExtension "Via1" "Metal1"
    'cutClass "VA"
    'width 0.2
    'concaveCornerLength 0.06
    0.09
  )
) ;orderedSpacings
```

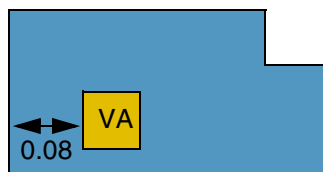
Via1
 Metal1



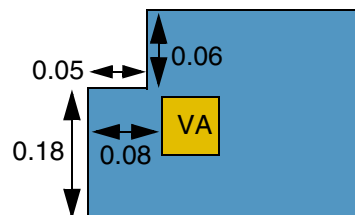
a) FAIL. The end-of-line width is 0.18 (< 0.2) and the lengths of the concave sides are 0.04 and 0.05 (both less than 0.06). However, the Metal1 extension is only 0.08 (< 0.09).



b) PASS. The end-of-line width is 0.18 (< 0.2) and the lengths of the concave sides are 0.04 and 0.05 (both less than 0.06). The Metal1 extension is 0.09.



c) The constraint does not apply because there is no concave corner touching the end-of-line edge that extends over the via cut.



d) The constraint does not apply because the length of a concave edge is equal to 0.06 (and not less than 0.06).

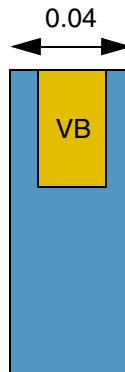
Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 6: minEndOfLineEdgeExtension with cutClass, width, minWidth, and shortEdgeOnly

The short edge of a rectangular cut of type VB must have an extension of at least 0.02 on an end-of-line edge that has width greater than or equal to 0.05 and less than 0.1.

```
orderedSpacings (  
  ( minEndOfLineEdgeExtension "Via1" "Metal1"  
    'cutClass "VB"  
    'width 0.1 'minWidth 0.05  
    'shortEdgeOnly  
    0.02  
  )  
) ;orderedSpacings
```



FAIL. The constraint does not apply because the end-of-line width is only 0.04 (<0.05).

Virtuoso Technology Data Constraints Reference

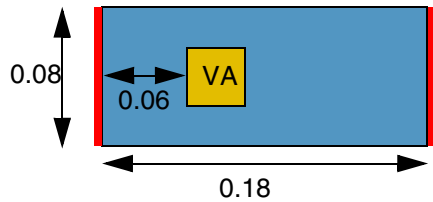
Extension Constraints

Example 7: minEndOfLineEdgeExtension with minVoltage and maxVoltage

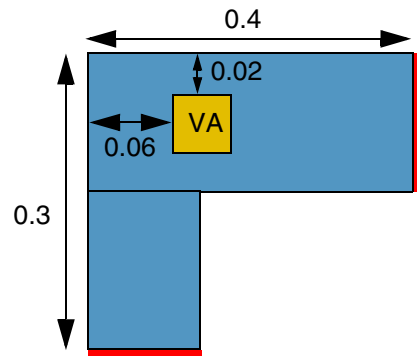
A cut of type VA must have an extension of at least 0.12 on an end-of-line edge less than 0.2 wide. Additionally, the minimum voltage on the shape must be greater than or equal to 1 and the maximum voltage less than or equal to 1.5.

Assume that the shapes have voltage 1.2.

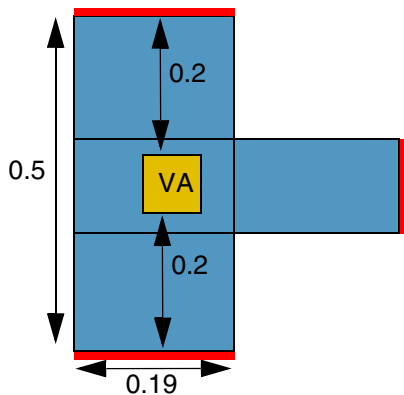
```
orderedSpacings (
  ( minEndOfLineEdgeExtension "Via1" "Metal1"
    'cutClass "VA"
    'width 0.2
    'equalRectWidth
    'minVoltage 1
    'maxVoltage 1.5
    0.12
  )
);orderedSpacings
```



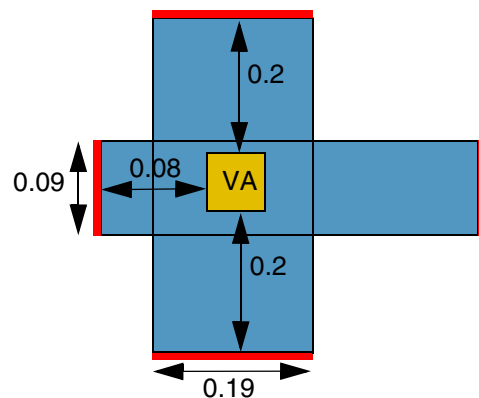
a) FAIL. The constraint applies but fails because the end-of-line extension is 0.06, which is less than 0.12. The equalRectWidth parameter ensures that only the short edge of the wire is considered for end-of-line.



b) The constraint does not apply on joints because both merged edges of joints were not end-of-lines.



c) PASS. The horizontal edge is considered end-of-line. The constraint applies and passes because the extension is 0.2, which is more than 0.12.



d) FAIL. Each red edge is considered end-of-line. The constraint applies and fails because the horizontal extension is 0.08, which is less than 0.12.

minEndOfLineExtension

```
orderedSpacings (
    ( minEndOfLineExtension tx_layer1 tx_layer2
      (
        f_eolWidth
        f_eolWithin
        [f_eolSpace [f_parWithin [f_parSpace]]]
        f_extension
      )
    )
) ;orderedSpacings
```

Specifies the minimum extension of an end-of-line edge of a *layer1* shape past a *layer2* shape when three parallel *layer1* edges are present on three sides of the enclosing *layer1* shape at a distance less than or equal to the specified values.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. The shape on this layer encloses a shape on <i>tx_layer2</i> . Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. The shape on this layer is enclosed. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension of an end-of-line edge past a cut edge must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

f_eolWidth

The constraint applies only if the end-of-line width is less than this value.

f_eolSpace f_eolWithin

These two parameters together define a verification region that extends beyond and on both sides of the end-of-line edge up to a distance equal to *eolSpace* and *eolWithin*, respectively. The constraint applies only if a *layer1* neighboring edge is present inside this verification region (and inside the verification regions defined by *parWithin* and *parSpace*).

f_parWithin f_parSpace

These two parameters together define verification regions on either side of the enclosing *layer1* shape. The verification regions extend up to a distance equal to *parWithin* along the length of the enclosing *layer1* shape and equal to *parSpace* on both sides of the end-of-line edge. The constraint applies only if a *layer1* edge is present inside both verification regions (and inside the verification region defined by *eolSpace* and *eolWithin*).

Virtuoso Technology Data Constraints Reference

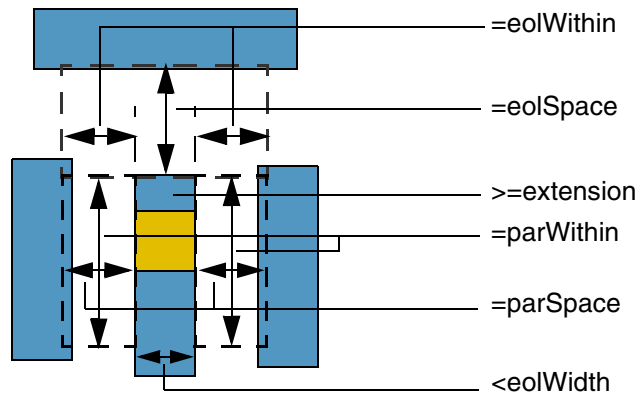
Extension Constraints

Example

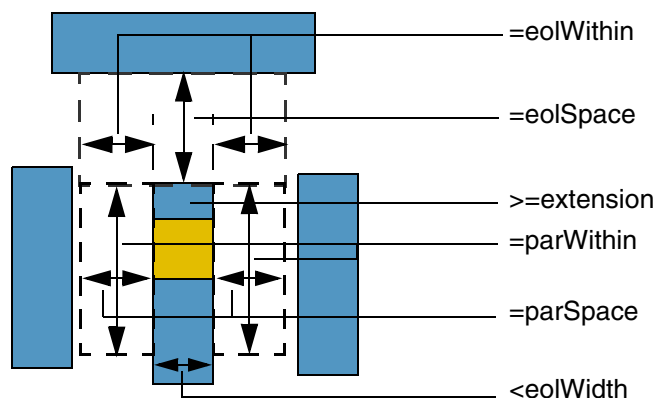
The extension of a Metal1 end-of-line edge past a Via1 cut edge must be greater than or equal to "extension" if three parallel Metal1 edges are present on three sides of the enclosing Metal1 shape.

```
orderedSpacings (
  ( minEndOfLineExtension "Metal1" "Via1"
    (
      eolWidth
      eolWithin
      eolSpace parWithin parSpace
      extension
    )
  )
);orderedSpacings
```

Via1
 Metal1
 Verification region



a) PASS. Metal1 edges are present inside the three verification regions and the extension is greater than or equal to *extension*.



b) The constraint does not apply because Metal1 edges are not present in all three verification regions.

minExtensionDistance

```
orderedSpacings (
  ( minExtensionDistance tx_layer1 tx_layer2
    ['horizontal | 'vertical]
    ['layer2Voltage f_layer2Voltage]
    ['cutClass {f_width | (f_width f_length) | t_name}]
    ['exceptEdgeLengthRange g_exceptEdgeLengthRange]
    ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    f_extension
    ['coincidentAllowed] ['manhattan]
  )
) ;orderedSpacings

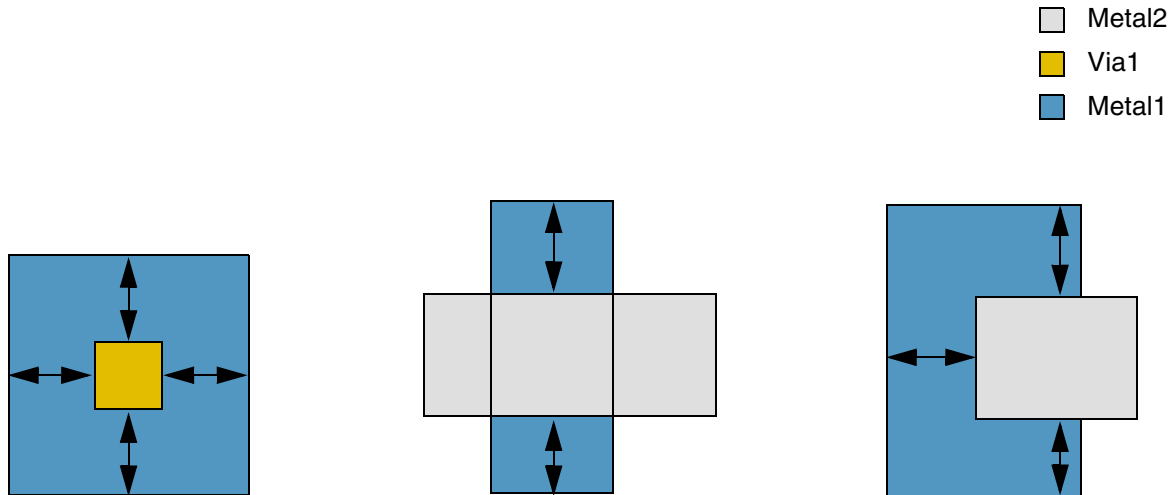
spacingTables (
  ( minExtensionDistance tx_layer1 tx_layer2
    (( "width" nil nil )
      ['horizontal | 'vertical]
      ['layer2Voltage f_layer2Voltage]
      ['cutClass {f_width | (f_width f_length) | t_name}]
      ['exceptEdgeLengthRange g_exceptEdgeLengthRange]
      ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
    )
    (g_table)
    ['coincidentAllowed] ['manhattan]
  )
) ;spacingTables
```

Specifies the minimum extension of a shape on *layer1* past a shape on *layer2*. The minimum extension is optionally dependent on the length of the *layer2* shape.

Virtuoso Technology Data Constraints Reference

Extension Constraints

This is an ordered constraint, which means that the minimum extension of a *layer1* shape past a *layer2* shape is not the same as the minimum extension of that *layer2* shape past the *layer1* shape.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension must be greater than or equal to this value.
"width" nil nil	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Extension Constraints

g_table The *g_table* row is defined as:

(f_width f_extension)

where,

- *f_width* is the width of the *layer1* shape. The width of the *layer1* shape must be greater than or equal to this value for the corresponding extension value to apply.
- *f_extension* is the extension of the *layer1* shape past the *layer2* shape. The extension must be greater than or equal to this value.

Type: A 1-D table specifying floating-point width and extension values

Parameters

'horizontal | 'vertical

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to the extensions measured in this direction. If direction is not specified, the extension is measured in any direction.

Type: Boolean

'layer2Voltage *f_layer2Voltage*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if the voltage for *layer2* shape is greater than or equal to this value.

'cutClass {*f_width* | (*f_width f_length*) | *t_name*}

(Virtuoso Advanced Node for Layout Standard) The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

'exceptEdgeLengthRange *g_exceptEdgeLengthRange*

Virtuoso Technology Data Constraints Reference

Extension Constraints

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if the length of the *layer2* shape falls in this range.

Type: Floating-point values specifying a range of lengths that are exempted

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)`
`| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)`

(Virtuoso Advanced Node for Layout Standard) Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- `'outsideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

(Virtuoso Advanced Node for Layout Standard) The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

(Virtuoso Advanced Node for Layout Standard) The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'coincidentAllowed` The edges of an enclosed shape can coincide with the edges of the enclosing shape. Otherwise, the enclosed shape must meet the extension requirement.

Type: Boolean

`'manhattan` The constraint uses Manhattan distance, which allows a larger spacing at the corners.

By default, the constraint uses Euclidean measurement.

Type: Boolean

Virtuoso Technology Data Constraints Reference

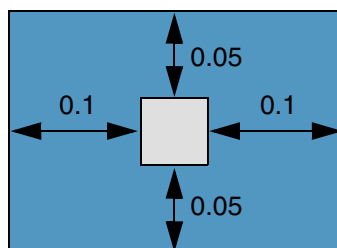
Extension Constraints

Example

The horizontal extension of a Metal1 shape past a Metal2 shape must be at least 0.1.

```
orderedSpacings (  
  ( minExtensionDistance "Metal1" "Metal2"  
    'horizontal  
    0.1  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



a) PASS. The extension of the Metal1 shape past the Metal2 shape in the horizontal direction is 0.1.



b) The constraint does not apply because the Metal2 shape does not have Metal1 extensions in the horizontal direction.

Virtuoso Technology Data Constraints Reference

Extension Constraints

minExtensionEdge

```
spacings (
  ( minExtensionEdge tx_layer1 tx_layer2
    ['cutClass {f_width | (f_width f_length) | t_name}]
    ['includeCorner]
    'width f_width ['maxWidth f_maxWidth]
    'paraLength f_paraLength
    ['within {f_paraWithin | (f_minWithin f_maxWithin)}]
    ['paraEdgeWidth f_paraWidth]
    ['exceptExtraCut ['distanceWithin f_distance]]
    ['exceptTwoEdges ['exceptWithin f_exceptWithin]]
    f_extension
  )

  ( minExtensionEdge tx_layer1 tx_layer2
    ['cutClass {f_width | (f_width f_length) | t_name}]
    ['includeCorner]
    'minSpanLength f_spanLength
    ['maxSpanLength f_maxSpanLength]
    ['paraLength f_paraLength]
    ['within {f_paraWithin | (f_minWithin f_maxWithin)}]
    ['paraEdgeWidth f_paraWidth]
    ['exceptExtraCut ['distanceWithin f_distance]]
    ['exceptTwoEdges ['exceptWithin f_exceptWithin]]
    f_extension
  )

  ( minExtensionEdge tx_layer1 tx_layer2
    ['cutClass {f_width | (f_width f_length) | t_name}]
    ['includeCorner]
    'convexLength f_convexLength
      'adjacentLength f_adjLength
      'minLength f_minLength
      'paraWithin f_paraWithin
    f_extension
  )

  ( minExtensionEdge tx_layer1 tx_layer2
    ['cutClass {f_width | (f_width f_length) | t_name}]
    'twoSides
      ['eolWidth f_eolWidth]
      ['exceptConcaveCorner f_exceptConcaveCorner]
      ['cutToMetalSpace f_cutToMetalSpace
        ['layer f_layer] ['edgeExtension f_edgeExtension]
      ]
      ['horizontal | 'vertical]
      ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
      'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  )
)
```

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
        ]  
        f_extension  
    )  
) ; spacings
```

Specifies the minimum extension of a shape on one layer past a shape on another layer. Typically, *layer1* is a metal layer and *layer2* is a cut layer. The *layer1* edge that must satisfy the constraint is determined by any parallel neighboring edges that are present.

The extension value specified with this constraint is typically larger than the value specified using `minExtensionDistance` or `minOppExtension` constraints and applies if the metal shape enclosing the cut shape has width greater than *minWidth* and is less than *parWithin* away from a neighboring shape whose parallel run length with the enclosing metal shape is greater than *parLength*, as shown in the figure below. Additionally, the cut shape must have a parallel run length greater than zero with the neighboring shape.

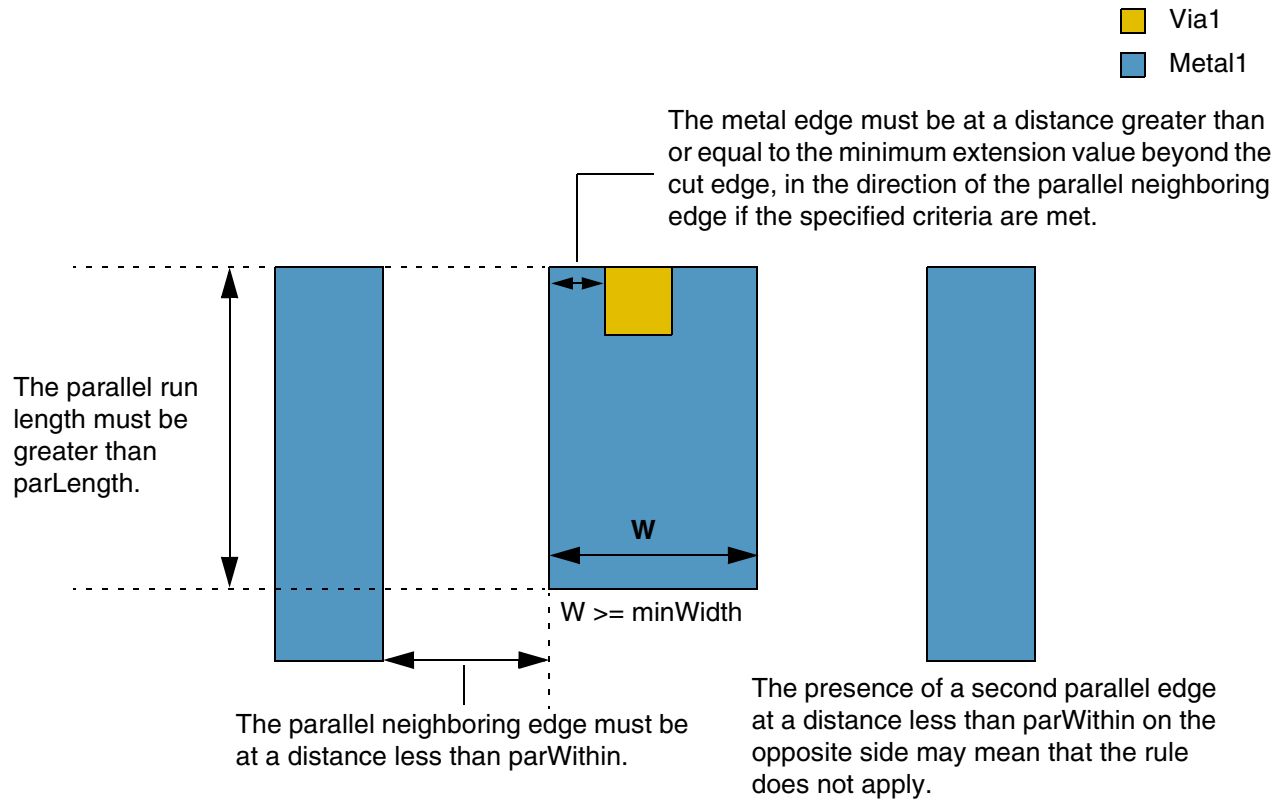
The constraint may not apply if:

- An extra cut shape exists in the same metal extension.
- The enclosing metal shape has two neighboring shapes, on opposite sides, less than *parWithin* away and with parallel run length greater than *parLength*.

Virtuoso Technology Data Constraints Reference

Extension Constraints

- The cut shape belongs to a class different from the one specified.

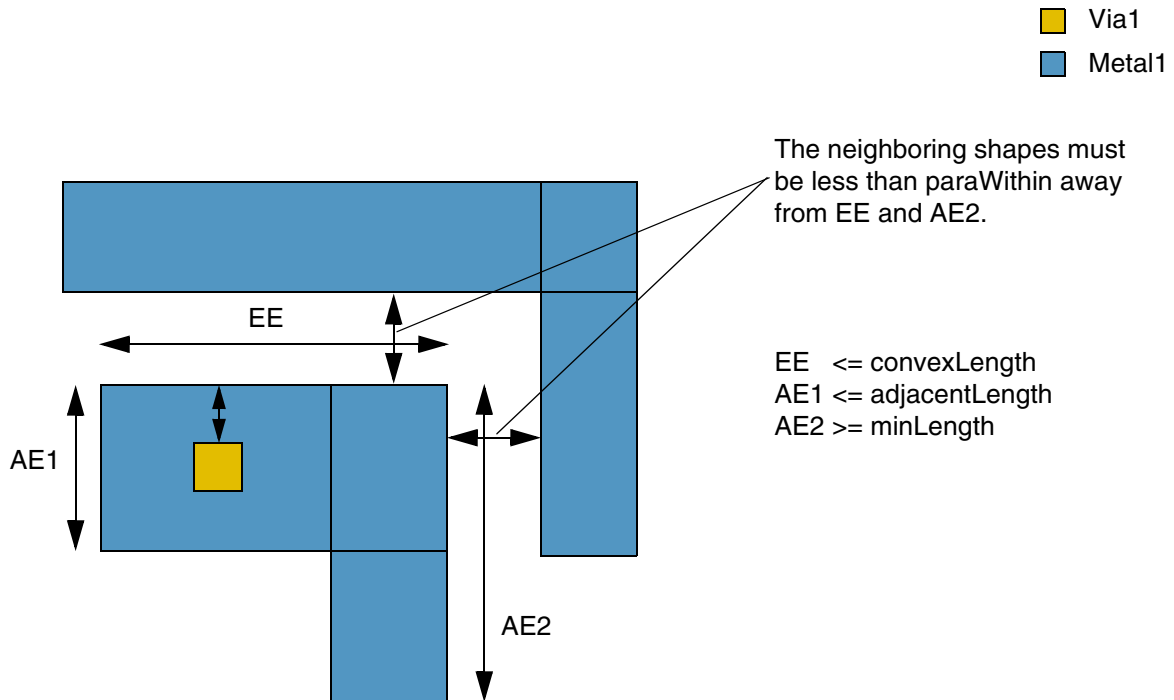


In some processes, the conditions illustrated in the figure below must be satisfied for the constraint to apply. The enclosing edge (EE) and an adjacent edge (AE1) must lie between

Virtuoso Technology Data Constraints Reference

Extension Constraints

two convex corners. Additionally, the cut must have parallel run length greater than 0 with neighboring shapes that are parallel to EE and the other adjacent edge (AE2).



Values

<i>tx_layer1</i>	The first layer (metal) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer (cut) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_extension</i>	The minimum extension value that must be satisfied.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

'cutClass {*f_width* | (*f_width* *f_length*) | *t_name*}

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

'width *f_width*

The width of the enclosing *layer1* shape must be greater than or equal to this value.

'maxWidth *f_maxWidth*

The width of the enclosing *layer1* shape must be less than this value.

'within {*f_parWithin* | (*g_minWithin* *g_maxWithin*)}

The constraint applies if the distance between the enclosing *layer1* edge and a parallel metal edge is less than *parWithin*, or if the distance between the two edges is greater than or equal to *minWithin* and less than *maxWithin*.

'paraLength *f_paraLength*

The constraint applies only if the parallel run length between the enclosing *layer1* shape and a neighboring edge is greater than this value (*paraLength*).

'paraEdgeWidth *f_paraEdgeWidth*

The constraint applies only if the width of the parallel neighboring shape is greater than or equal to this value.

'exceptExtraCut

The constraint does not apply if there exists another *layer2* shape in the same metal intersection.

Type: Boolean

'distanceWithin *f_distanceWithin*

If there are two *layer2* shapes in the same metal intersection, then they must be at a distance greater than this value for the constraint to apply.

Virtuoso Technology Data Constraints Reference

Extension Constraints

'exceptTwoEdges	The constraint does not apply if neighboring edges with parallel run length greater than <i>parLength</i> exist on opposite sides of the enclosing metal shape at a distance less than <i>parWithin</i> from it. Type: Boolean
'exceptWithin <i>f_exceptWithin</i>	The constraint applies if the neighboring edges on opposite sides are at a distance less than this value.
'includeCorner	The constraint applies to the corner of the cut shape, using Euclidean measurement. This check is required if the cut shape is outside the projection of the neighboring shape. Type: Boolean
'minSpanLength <i>f_minSpanLength</i>	The constraint applies only if the span length of the metal shape in the direction perpendicular to the parallel run length is greater than or equal to this value.
'maxSpanLength <i>f_maxSpanLength</i>	The constraint applies only if the span length of the metal shape in the direction perpendicular to the parallel run length is less than this value.
'twoSides	(Virtuoso Advanced Node for Layout Standard) If an edge of a <i>layer2</i> shape has a <i>layer1</i> extension less than or equal to the constraint value, then the entire opposite edge of the <i>layer2</i> shape must also have an extension less than or equal to the constraint value. Type: Boolean
'eolWidth <i>f_eolWidth</i>	The constraint applies only if the end-of-line width of the metal edges satisfying the 'twoSides extension requirement is greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'exceptConcaveCorner f_exceptConcaveCorner`

(Virtuoso Advanced Node for Layout Standard) The distance of a concave corner from a cut edge orthogonal to the cut edges satisfying the `'twoSides` extension requirement must be greater than this value. The distance to the concave corner is measured after projecting the extension value on the orthogonal bottom edge.

`'cutToMetalSpace f_cutToMetalSpace`

The distance of a cut edge satisfying the `'twoSides` extension requirement to a different-net wire must be greater than or equal to this value.

`'layer tx_layer`

(Virtuoso Advanced Node for Layout Only) The metal layer to which `'cutToMetalSpace` spacing must be measured.

Type: String (layer name) or Integer (layer number)

`'edgeExtension f_edgeExtension`

(Virtuoso Advanced Node for Layout Only) The cut to different-net spacing of a cut edge satisfying the `'twoSides` extension requirement is measured after projecting this extension value on the metal edge in the direction of the cut edge.

`'convexLength f_convexLength`

The constraint applies only if the length of the enclosing edge (EE) is less than or equal to this value.

In addition, the `'adjacentLength`, `'minLength` and `'paraWithin` parameters must be specified and met.

`'adjacentLength f_adjLength`

The constraint applies only if the length of the adjacent edge (AE1) lying between two convex corners is less than or equal to this value.

`'minLength f_minLength`

The constraint applies only if the length of the other adjacent edge (AE2) is greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'paraWithin f_paraWithin`

The constraint applies if the distance of both the enclosing edge (EE) and the other adjacent edge (AE2) from its neighboring edge is less than this value.

`'horizontal | 'vertical`

(Virtuoso Advanced Node for Layout Standard) The direction in which the extension is measured.

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
```

(Virtuoso Advanced Node for Layout Standard) Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- 'outsideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Standard) The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Standard) The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Examples

- [Example 1: minExtensionEdge with width, paraLength, and within](#)
- [Example 2: minExtensionEdge with exceptTwoEdges](#)
- [Example 3: minExtensionEdge with paraEdgeWidth and includeCorner](#)
- [Example 4: minExtensionEdge with twoSides and eolWidth](#)
- [Example 5: minExtensionEdge with exceptConcaveCorner and cutToMetalSpace](#)

Virtuoso Technology Data Constraints Reference

Extension Constraints

- [Example 6: minExtensionEdge with cutToMetalSpace, layer, and edgeExtension](#)
- [Example 7: minExtensionEdge with minSpanLength](#)
- [Example 8: minExtensionEdge with convexLength](#)

Virtuoso Technology Data Constraints Reference

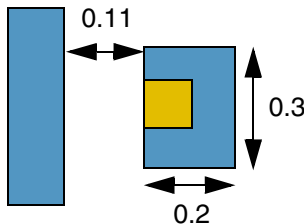
Extension Constraints

Example 1: *minExtensionEdge* with *width*, *paraLength*, and *within*

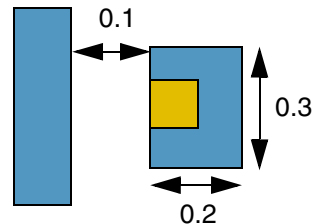
The extension of a Metal1 shape with minimum width 0.2 past a Via1 cut must be at least 0.05 in the direction of a neighboring Metal1 edge less than 0.11 away, when the parallel run length for the Metal1 shapes is greater than 0.25.

```
spacings(  
  ( minExtensionEdge "Metal1" "Via1"  
    'width 0.2  
    'paraLength 0.25  
    'within 0.11  
    0.05  
  )  
); spacings
```

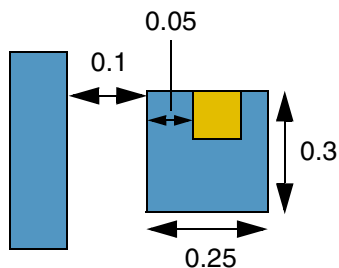
■ Via1
■ Metal1



a) The constraint does not apply because the distance between the parallel neighboring shape and the enclosing shape is 0.11. For the constraint to apply, this distance must be less than 0.11.



b) FAIL. The extension of the left metal edge, to which the constraint applies, past the cut shape is 0 (< 0.05).



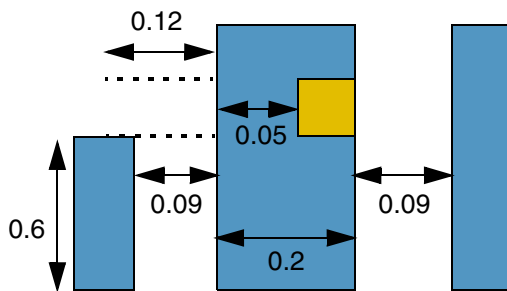
c) PASS. The extension of the metal edge, to which the constraint applies, past the cut shape is 0.05.

Example 2: minExtensionEdge with exceptTwoEdges

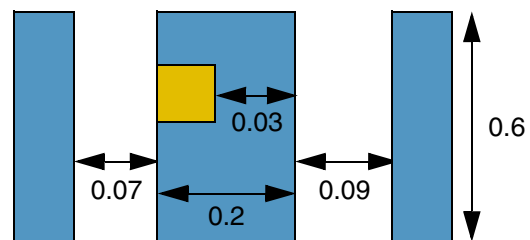
The extension of a Metal1 shape with minimum width 0.2 past a Via1 cut must be at least 0.05 in the direction of a neighboring Metal1 edge greater than or equal to 0.08 and less than 0.1 away from it, when the parallel run length for the Metal1 shapes is greater than 0.5. The constraint does not apply if such neighboring shapes are present on both sides, except when their distance from the enclosing shape is less than 0.12.

```
spacings (
  ( minExtensionEdge "Metal1" "Via1"
    'width 0.2
    'paraLength 0.5
    'within (0.08 0.1)
    'exceptTwoEdges exceptWithin 0.12
    0.05
  )
) ;spacings
```

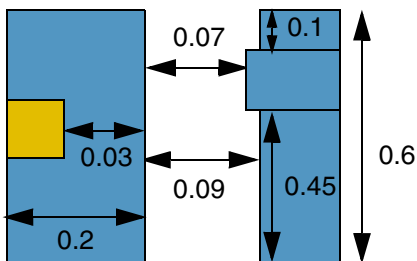
■ Via1
■ Metal1



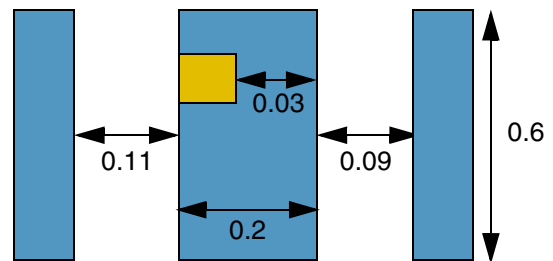
a) FAIL. The right neighbor does not have a left neighbor in the dotted region, so 'exceptTwoEdges is not triggered. This implies that there must be a 0.05 enclosure on the right.



b) FAIL. The distance of the left neighbor is 0.07 (<0.08). Therefore, 'exceptTwoEdges is not triggered. The enclosure on the right is 0.03, which is less than the required value (<=0.05).



c) The constraint does not apply. The portion of the neighboring shape at a distance of 0.07 is ignored. The other two portions are treated individually and both have parallel run length less than 0.5.



d) The constraint does not apply. 'exceptTwoEdges is triggered and both neighbors are at a distance less than 0.12 from the enclosing shape. A violation would occur if 'exceptWithin was not defined.

Virtuoso Technology Data Constraints Reference

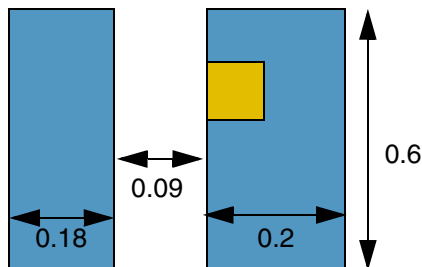
Extension Constraints

Example 3: minExtensionEdge with paraEdgeWidth and includeCorner

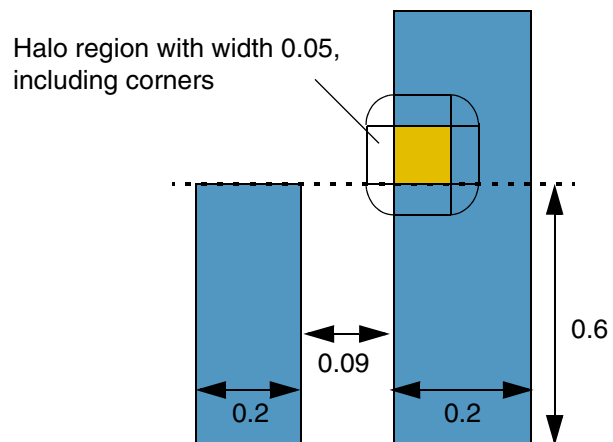
The extension of a Metal1 shape with minimum width 0.2 past a Via1 cut must be at least 0.05, for edges as well as corners, in the direction of a neighboring Metal1 edge less than 0.1 away from it, when the parallel run length for the Metal1 shapes is greater than 0.5. The width of the neighboring shape must be at least 0.2.

```
spacings(  
  ( minExtensionEdge "Metal1" "Via1"  
    'width 0.2  
    'paraLength 0.5  
    'paraEdgeWidth 0.2  
    'within 0.1  
    'includeCorner  
    0.05  
  )  
); spacings
```

■ Via1
■ Metal1



a) The constraint does not apply because the width of the neighboring shape is 0.18 (<0.2). The constraint would fail if 'paraEdgeWidth was not specified because the via cut does not have a 0.05 extension on the left.



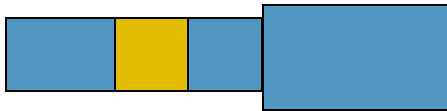
b) FAIL. The lower-left corner of the halo region overlaps the projection of the neighboring shape. Had 'includeCorner not been specified, the constraint would have been met.

Example 4: *minExtensionEdge with twoSides and eolWidth*

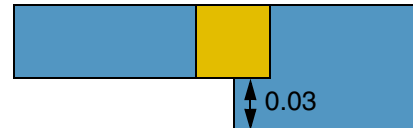
The extension of a Metal1 shape past a Via1 cut must be at least 0.02 along the entire length of a pair of opposite edges of the cut shape, and the end-of-line width of the metal edges that satisfy the 'twoSides requirement must be at least 0.15.

```
spacings (
  ( minExtensionEdge "Metal1" "Via1"
    'twoSides
      eolWidth 0.15
      0.02
  )
) ;spacings
```

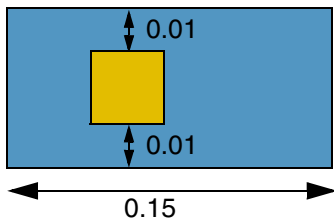
■ Via1
■ Metal1



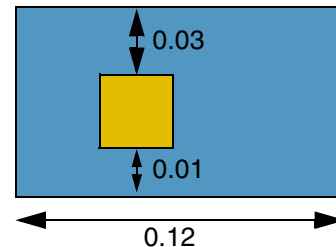
a) PASS. The metal shape has an extension of 0 along the entire length of a pair of opposite edges of the cut shape.



b) FAIL. The metal shape has an extension of 0.03 (>0.02) on a portion of the bottom edge of the cut shape.



c) PASS. The metal shape has an extension of 0.01 (<0.02) along the entire length of a pair of opposite edges of the cut shape. The end-of-line width requirement is also satisfied.



d) The constraint does not apply because the end-of-line width is less than 0.15.

Virtuoso Technology Data Constraints Reference

Extension Constraints

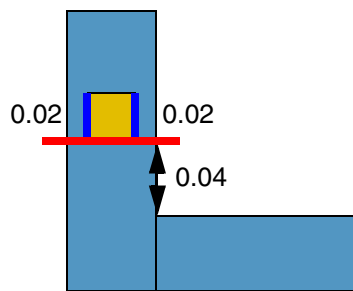
Example 5: *minExtensionEdge* with *exceptConcaveCorner* and *cutToMetalSpace*

The extension of a Metal1 shape past a Via1 cut must be at least 0.02 along the entire length of a pair of opposite edges of the cut shape. The following conditions must also be satisfied:

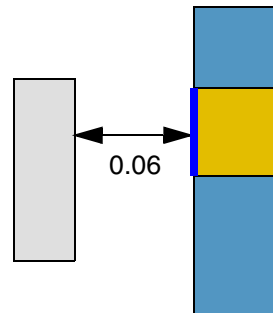
- The distance of a metal concave corner from a cut edge orthogonal to the cut edges satisfying the extension requirement must be greater than 0.05.
- The distance of a cut edge, satisfying the extension requirement, from a different-net wire must be greater than or equal to 0.07.

```
spacings (
  ( minExtensionEdge "Metal1" "Via1"
    'twoSides
      exceptConcaveCorner 0.05
      'cutToMetalSpace 0.07
    0.02
  )
) ;spacings
```

□ Metal2
■ Via1
■ Metal1



a) FAIL. The blue cut edges fulfill the 0.02 extension requirement, but after projecting the extension value of 0.02 (in red) on the orthogonal bottom edge, the distance of the concave corner from this cut edge is 0.04 (<0.05).



b) FAIL. The blue cut edges fulfill the 0.02 extension requirement, but the distance of one such edge from a different-net wire is 0.06 (<0.07).

Example 6: minExtensionEdge with cutToMetalSpace, layer, and edgeExtension

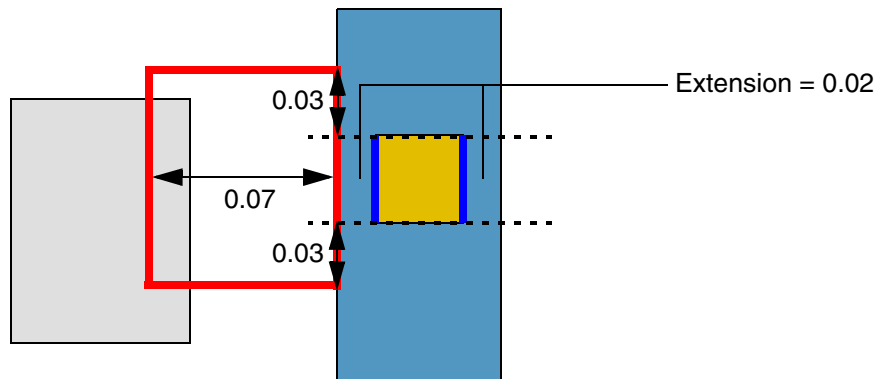
The extension of a Metal1 shape past a Via1 cut must be at least 0.02 along the entire length of a pair of opposite edges of the cut shape, and the distance of a cut edge, satisfying the extension requirement, from a different-net wire must be greater than or equal to 0.07. Additionally, this distance must be measured after projecting the metal edge by 0.03 along the direction of the cut.

```

spacings (
  ( minExtensionEdge "Metal1" "Via1"
    'twoSides
    'cutToMetalSpace 0.07
    'layer "Metal2" 'edgeExtension 0.03
    0.02
  )
) ;spacings

```

Metal2
 Via1
 Metal1



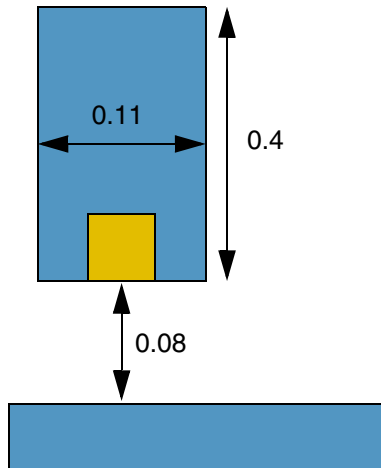
FAIL. The blue cut edges fulfill the 0.02 extension requirement, but the distance measured from Metal1 edge to the Metal2 object, after applying an extension of 0.03 on the Metal1 edge along the direction of the cut edge, is less than 0.07. In other words, no neighbor must lie in the region defined by the red boundary.

Example 7: *minExtensionEdge* with *minSpanLength*

The extension of a Metal1 shape with minimum span length 0.2 past a Via1 cut must be at least 0.02 in the direction of a neighboring Metal1 edge less than 0.09 away, when the parallel run length for the Metal1 shapes is greater than 0.01.

```
spacings (
  ( minExtensionEdge "Metal1" "Via1"
    'paraLength 0.1
    'within 0.09
    'minSpanLength 0.2
    0.02
  )
) ;spacings
```

■ Via1
■ Metal1



FAIL. The constraint applies because the span length of the metal shape in the direction perpendicular to the parallel run length ($0.11 > 0.1$) is 0.4 (≥ 0.2). However, the extension of the bottom metal edge, to which the constraint applies, past the cut shape is 0 (< 0.02).

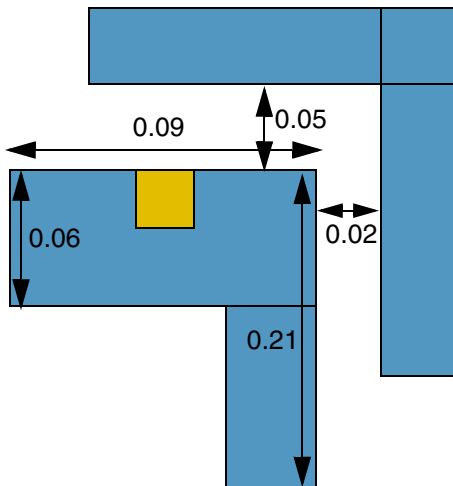
Example 8: *minExtensionEdge* with *convexLength*

The extension of a Metal1 edge—enclosing edge (EE)—past a Via1 cut must be at least 0.02 if the following conditions are true:

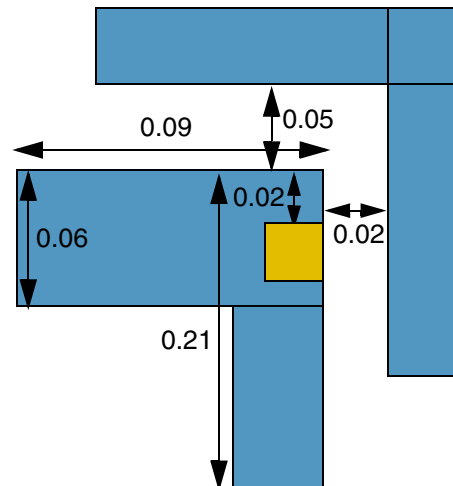
- EE is less than or equal to 0.1 and lies between two convex corners.
- EE has two adjacent edges, one of which (AE1) also lies between two convex corners and is less than or equal to 0.08. The other adjacent edge (AE2) is greater than or equal to 0.2.
- The distance of EE and AE2 from a neighboring edge is less than 0.09.

```
spacings (
  ( minExtensionEdge "Metal1" "Via1"
    'convexLength 0.1
    'adjacentLength 0.08
    'paraWithin 0.09
    'minLength 0.2
    0.02
  )
) ;spacings
```

■ Via1
■ Metal1



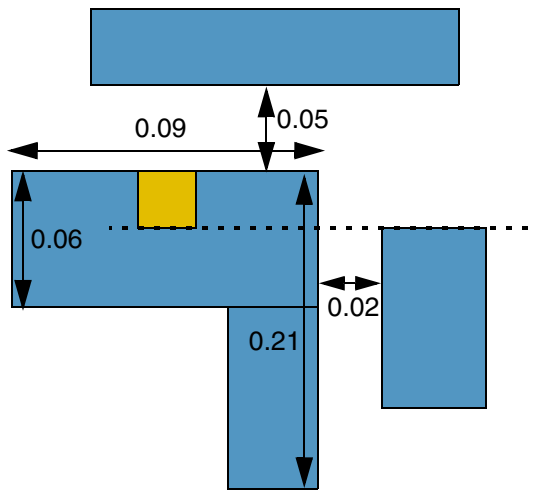
a) FAIL. All conditions are satisfied, but the extension of the enclosing edge on the top edge of the cut shape is 0.



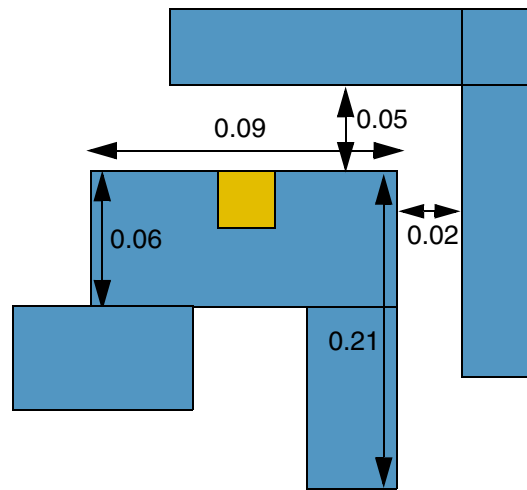
b) PASS. The extension of the enclosing edge on the top edge of the cut shape is 0.02.

Virtuoso Technology Data Constraints Reference

Extension Constraints



c) The constraint does not apply because the parallel run length of the cut shape with the neighboring shape is 0.



d) The constraint does not apply because the adjacent edge measuring 0.06 does not lie between convex corners.

minExtensionEolSpacing

```
spacingTables (
    ( minExtensionEolSpacing tx_metalLayer tx_cutLayer
      'width g_width
      ['cutClass {g_cutWidth | (g_cutWidth g_cutLength) | t_name}
      ['mask1 | 'mask2 | 'mask3 | 'mask4]
      ['cutWithin g_cutWithin]
      ['exceptNoCutInOtherMetalLength (g_cutDistance g_length)]
      (g_extensionEolSpacingTable)
    )
) ;spacingTables
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies a minimum cut edge extension when it is next to an end-of-line edge with spacing less than *g_extensionEolSpacingTable*.

Values

<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_extensionEolSpacingTable</i>	The table row is defined as (<i>f_spacing</i> <i>f_enclosure</i>) where: ■ <i>f_spacing</i>: Spacing of the end-of-line edges on layer1. ■ <i>f_enclosure</i>: The required enclosure of the cut on layer2 when the end-of-line spacing is greater than <i>f_spacing</i>. Type: A one-dimensional table specifying floating-point spacing and enclosure values.

Parameters

<i>'width g_width</i>	The constraint applies only if the wire width is less than the given value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
'cutClass {g_cutWidth | (g_cutWidth g_cutLength) | t_name}
```

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *g_width*: Width
- *g_length*: Length
- *t_name*: Name of the cut class

```
'mask1 | 'mask2 | 'mask3 | 'mask4
```

Specifies the mask a wire belongs to.

```
'cutWithin g_cutWithin
```

The constraint applies only if the triggering cut has a neighbor cut within the specified value.

```
'exceptNoCutInOtherMetalLength (g_cutDistance g_length)
```

The constraint does not apply if the neighbor wire has length greater than or equal to *g_length* and there is no cut on the neighbor wire that belongs to the specified cut class name if defined or any cut class having a distance to the triggering cut less than *g_cutDistance*.

Example

This example shows the required cut enclosure for cuts of cutClass VA on end-of-line wires of with lass than 0.1. If the end-of-line spacing is greater than 0.05 then the required cut enclosure is 0.07. If the end-of-line spacing is greater than 0.09 then the required cut enclosure is 0.06. If the end-of-line spacing is greater than 0.11 then the required cut enclosure is 0.09.

```
spacingTables (
  ; ( constraint          layer1          [layer2]
  ;   (( index1Definitions [index2Defintions]) [defaultValue] )
  ;   ( table) )
  ; ( -----)
  (minExtensionEolSpacing "Metall" "Vial"
    ("spacing" nil nil) 'width 0.1 'cutClass "VA")
    (0.05 0.07
     0.09 0.06
     0.11 0.09
    ) )
) ;spacingTables
```

minExtensionOnLongSide

```
orderedSpacings (
    ( minExtensionOnLongSide tx_layer1 tx_layer2
      ['widthRanges (g_ranges) ['otherWidthRanges (g_otherWidthRanges)]]
      f_extension
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum extension of a shape on *layer1* past the long edges of a shape on *layer2*.

Optionally, the constraint can be applied to shapes with certain widths. cat

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension must be greater than or equal to this value.

Parameters

'widthRanges g_ranges

The constraint applies only if the width of the *layer1* shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint

'otherWidthRanges g_otherWidthRanges

The constraint applies only if the width of the *layer2* shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint

Virtuoso Technology Data Constraints Reference

Extension Constraints

Examples

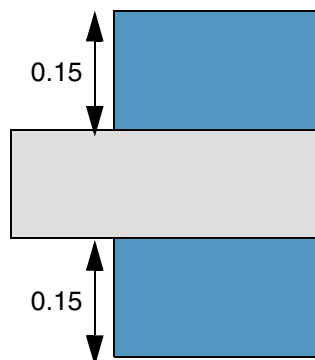
- Example 1: minExtensionOnLongSide
- Example 2: minExtensionOnLongSide with widthRanges and otherWidthRanges

Example 1: minExtensionOnLongSide

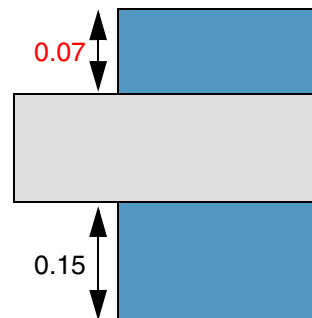
A Metal1 shape must have an extension greater than or equal to 0.15 on the long edges of a Metal2 shape.

```
orderedSpacings (  
  ( minExtensionOnLongSide "Metal1" "Metal2"  
    0.15  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



a) PASS. The Metal1 shape has 0.15 extension on both long edges of the Metal2 shape.



b) FAIL. The extension of the Metal1 shape past the top long edge of the Metal2 shape is only 0.07 (<0.15).

Virtuoso Technology Data Constraints Reference

Extension Constraints

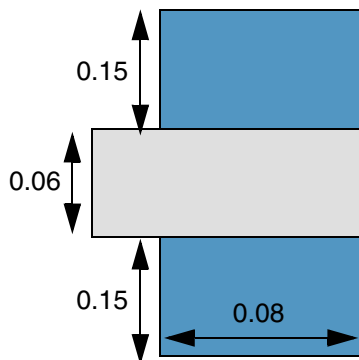
Example 2: *minExtensionOnLongSide* with *widthRanges* and *otherWidthRanges*

A Metal1 shape must have an extension greater than or equal to 0.15 on the long edges of a Metal2 shape if all of the following conditions are met:

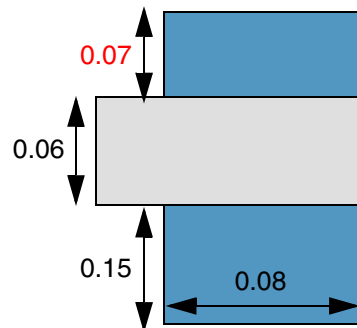
- Width of the Metal1 shape is greater than or equal to 0.08.
- Width of the Metal2 shape is either 0.04 or 0.06.

```
orderedSpacings (  
  ( minExtensionOnLongSide "Metal1" "Metal2"  
    'widthRanges (">= 0.08")  
    'otherWidthRanges (0.04 0.06)  
    0.15  
  )  
) ;orderedSpacings
```

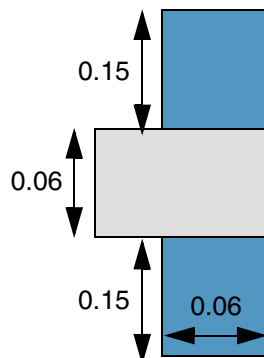
□ Metal2
■ Metal1



a) PASS. The Metal1 shape is 0.08 wide and the Metal2 shape is 0.06 wide. The Metal1 shape has 0.15 extension on both long edges of the Metal2 shape.



b) FAIL. The Metal1 shape is 0.08 wide and the Metal2 shape is 0.06 wide, but the extension of the Metal1 shape past the top long edge of the Metal2 shape is only 0.07 (<0.15).



c) The constraint does not apply because the width of the Metal1 shape is 0.06 (<0.08).

minExtensionToCenterLine

```
orderedSpacings (
    ( minExtensionToCenterLine tx_layer1 tx_layer2
      ['horizontal | 'vertical]
      ['widthRanges (g_ranges)]
      ['otherWidthRanges (g_otherRanges)]
      f_extension
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum extension of a shape on *layer1*, measured from the centerline of a shape on *layer2*.

In some process nodes, the extension of a shape on a middle-of-line (MOL) layer must be measured from the centerline of a shape on a poly layer.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The minimum required extension.

Parameters

<code>'horizontal 'vertical</code>	The direction in which the extension is measured. If direction is not specified, the extension is measured in any direction. Type: Boolean
<code>'widthRanges (g_ranges)</code>	The constraint applies only if the width of the shape on <i>layer1</i> is in this range. Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint

Virtuoso Technology Data Constraints Reference

Extension Constraints

'otherWidthRanges (*g_otherRanges*)

The constraint applies only if the width of the shape on *layer2* is in this range.

Type: Floating-point values specifying a range of widths that trigger the constraint

Examples

- Example 1: minExtensionToCenterLine
- Example 2: minExtensionToCenterLine with horizontal
- Example 3: minExtensionToCenterLine with widthRanges and otherWidthRanges

Virtuoso Technology Data Constraints Reference

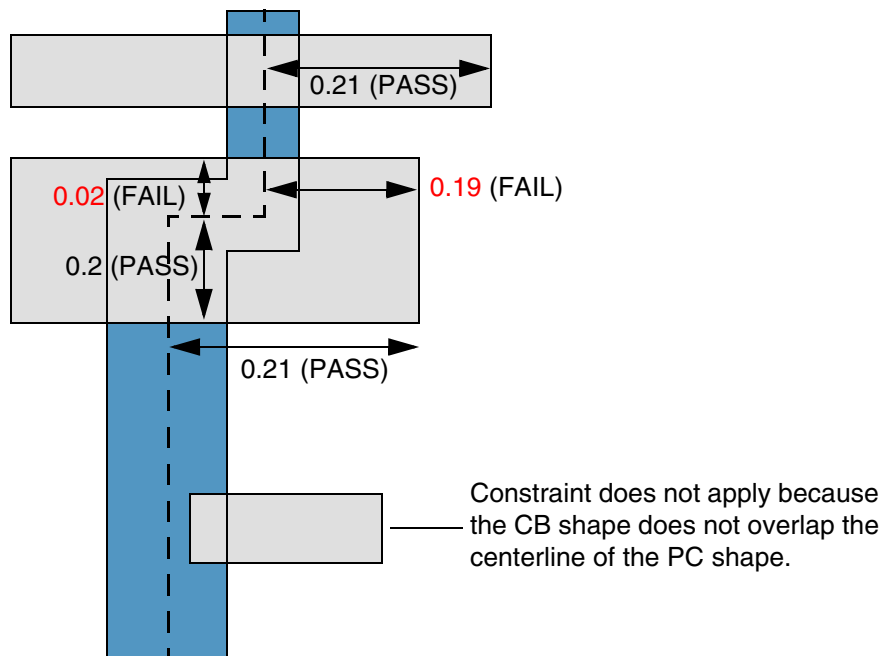
Extension Constraints

Example 1: *minExtensionToCenterLine*

The extension of a shape on layer CB from the centerline of a shape on layer PC must be at least 0.2 in any direction. This example assumes that all CB extensions to the left of the PC centerline are met.

```
orderedSpacings (
  ( minExtensionToCenterLine "CB" "PC"
    0.2
  )
) ;orderedSpacings
```

□ CB
■ PC



Virtuoso Technology Data Constraints Reference

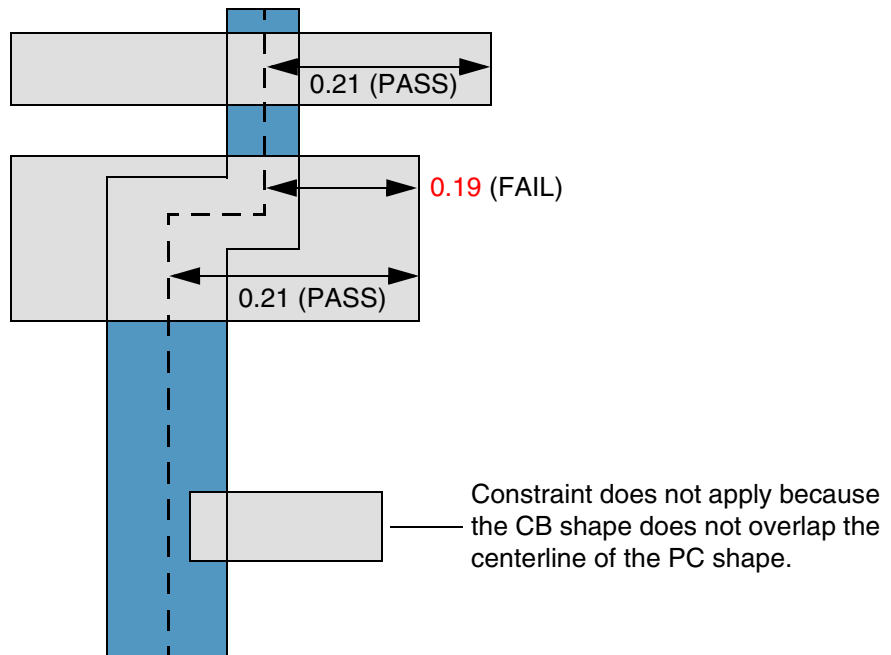
Extension Constraints

Example 2: *minExtensionToCenterLine* with horizontal

The extension of a shape on layer CB from the centerline of a shape on layer PC must be at least 0.2 in the horizontal direction. This example assumes that all CB extensions to the left of the PC centerline are met.

```
orderedSpacings (  
  ( minExtensionToCenterLine "CB" "PC"  
    'horizontal  
    0.2  
  )  
) ;orderedSpacings
```

□ CB
■ PC



Virtuoso Technology Data Constraints Reference

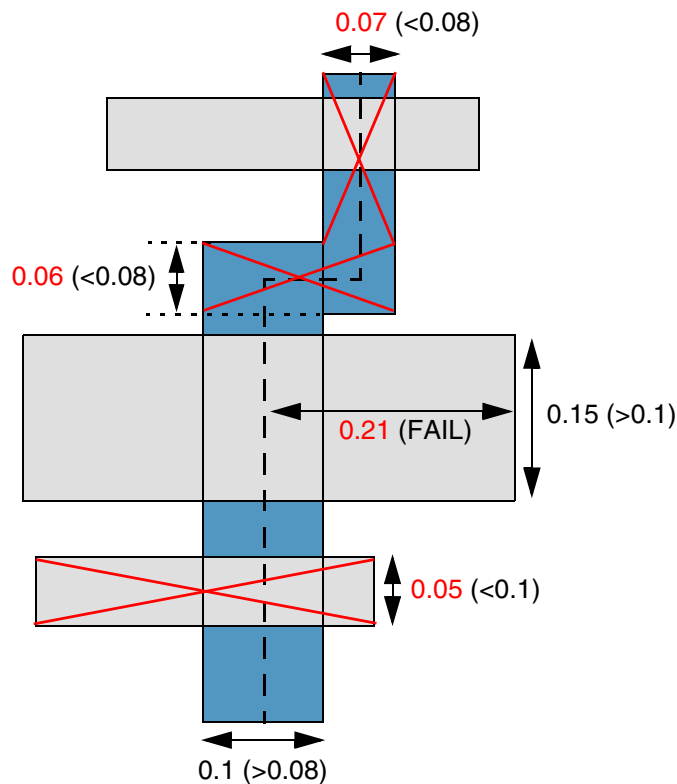
Extension Constraints

Example 3: *minExtensionToCenterLine* with *widthRanges* and *otherWidthRanges*

The extension of a shape on layer CB from the centerline of a shape on layer PC must be at least 0.22 in any direction. The width of the CB shape must be greater than 0.1 and the width of the PC shape must be greater than 0.08. This example assumes that all CB extensions to the left of the PC centerline are met.

```
orderedSpacings (  
  ( minExtensionToCenterLine "CB" "PC"  
    'widthRanges (">0.1")  
    'otherWidthranges (">0.08")  
    0.22  
  )  
) ;orderedSpacings
```

□ CB
■ PC



The constraint does not apply to regions where the width of the PC shape is less than or equal to 0.08 (denoted by the red cross). The constraint also does not apply if the width of a CB shape is less than or equal to 0.1.

minExtensionToCorner

```
orderedSpacings (
  ( minExtensionToCorner tx_layer1 tx_layer2
    { 'toConcaveCorner
      { ['parallelExtension f_parExt 'width f_width 'length f_length]
        | ['convexLength f_convexLength 'enclosure (f_encl1 f_encl2)]
      }
      | 'toConvexCorner ['eolWidth f_eolWidth] ['minLength f_minLength]
    }
    ['cutClass {f_width | (f_width f_length) | t_name}]
    ['viaEdgeType x_viaEdgeType]
    ['extendCornerBy f_extendCorner]
    ['exceptCornerTouch]
    f_spacing
  )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies how far a *layer2* shape must be from the corners of an enclosing *layer1* shape. The corners can be either concave or convex.

For convex corners, the constraint value is used to define a keepout region that the *layer2* shape must not overlap. This keepout region can be a triangle or a square depending on whether the 'eolWidth parameter is specified.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The value that determines how far a <i>layer2</i> shape must be from a corner of the enclosing <i>layer1</i> shape. The type of corner, concave or convex, determines how the spacing between the <i>layer2</i> shape and the corner is measured.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

'toConcaveCorner	The distance between an edge of a <i>layer2</i> shape and a concave corner of the enclosing <i>layer1</i> shape must be greater than or equal to the constraint value. Type: Boolean
'parallelExtension <i>f_parExt</i>	The concave corner is extended by the constraint value on both sides along the edge of the <i>layer1</i> shape perpendicular to the edge with width <i>width</i> to form a search window; the other side of the search window is equal to the width of the <i>layer1</i> shape. If a <i>layer2</i> shape is found inside the search window, then the <i>layer2</i> edge parallel to the <i>layer1</i> edge that forms the concave corner (and belongs to the <i>layer1</i> shape with width less than or equal to <i>width</i>) must have an enclosure greater than or equal to <i>parExt</i>. Enclosure is checked on a <i>layer2</i> edge only if that edge overlaps the search window.
'width <i>f_width</i>	The constraint applies only if the width of the enclosing <i>layer1</i> shape, an edge of which forms one part of the concave corner, is less than or equal to this value.
'length <i>f_length</i>	The constraint applies only if the length of the <i>layer1</i> edge that forms the other part of the concave corner is greater than or equal to this value.
'convexLength <i>f_convexLength</i>	The constraint applies to a concave corner only if one of the edges that forms the concave corner is less than this value in length and connects to a convex corner at the other end.
'enclosure (<i>f_encl1</i> <i>f_encl2</i>)	If an edge of a <i>layer2</i> shape has an enclosure less than <i>encl1</i> on an edge that forms a convex corner, the adjacent <i>layer2</i> edges must have enclosures greater than <i>encl2</i>.

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'toConvexCorner` A keepout region is defined in each convex corner of the shape on the first layer. If a shape on the second layer overlaps this keepout region, it is considered a violation.

- If both `'toConvexCorner` and `'eolWidth` are specified, the keepout region is a triangle with side equal to the constraint value or the smaller edge length.
- If only `'toConvexCorner` is specified, the keepout region is a square with sides equal to the constraint value.

Type: Boolean

`'eolWidth` *f_eolWidth*

The constraint does not apply if a convex corner is formed by an end-of-line edge with width less than this value. In other words, a triangular keepout region must not touch an end-of-line edge with width less than this value.

`'minLength` *f_minLength*

The constraint does not apply if both adjoining edges of an end-of-line edge are greater than or equal to this value.

In other words, an edge with width less than *eolWidth* does not qualify as an end-of-line edge if the length of any of its adjoining edges is less than this value.

`'cutClass` {*f_width* | (*f_width* *f_length*) | *t_name*}

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

`'viaEdgeType` *x_viaEdgeType*

The constraint applies only to the via cut edges of this type (the type is defined in the `viaEdgeType` constraint).

`'extendCornerBy` *f_extendCorner*

The constraint is applied after extending the corner by this value, both horizontally and vertically.

Virtuoso Technology Data Constraints Reference

Extension Constraints

'exceptCornerTouch The constraint does not apply if the *layer2* shape touches a corner.

Type: Boolean

Examples

- Example 1: minExtensionToCorner with toConvexCorner, eolWidth, and minLength
- Example 2: minExtensionToCorner with toConvexCorner
- Example 3: minExtensionToCorner with toConcaveCorner, parallelExtension, width, and length
- Example 4: minExtensionToCorner with toConcaveCorner, viaEdgeType, and exceptCornerTouch
- Example 5: minExtensionToCorner with toConcaveCorner, viaEdgeType, and exceptCornerTouch
- Example 6: minExtensionToCorner with toConcaveCorner and extendCornerBy

Virtuoso Technology Data Constraints Reference

Extension Constraints

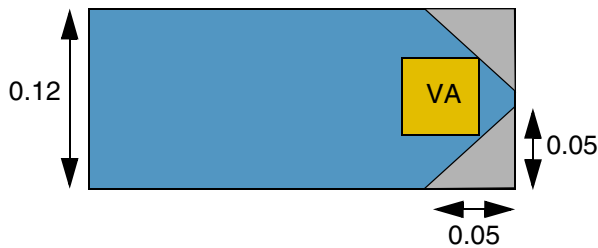
Example 1: *minExtensionToCorner* with *toConvexCorner*, *eoWidth*, and *minLength*

The distance of a VA via cut on layer Via1 from the convex corners of an enclosing Metal1 shape must be at least 0.05. The constraint does not apply if the convex corner is formed by an end-of-line edge less than 0.07 wide, whose both adjoining edges are greater than or equal to 0.05 long.

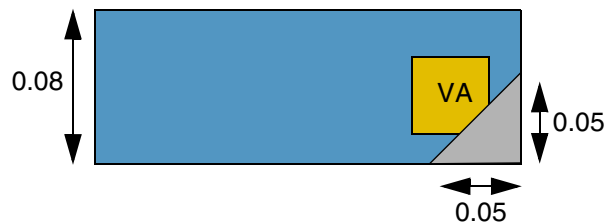
Note: Both 'toConvexCorner' and 'eoWidth' are specified; therefore, the keepout region is a triangle.

```
orderedSpacings (
  ( minExtensionToCorner "Metal1" "Via1"
    'toConvexCorner
    'eoWidth 0.07 'minLength 0.05
    'cutClass "VA"
    0.05
  )
) ;orderedSpacings
```

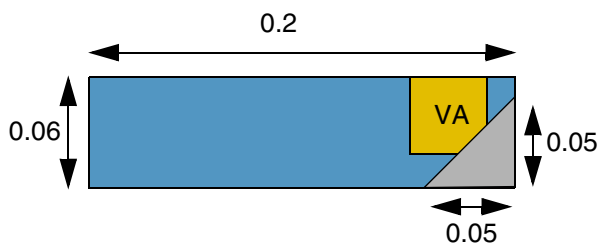
 Via1
 Metal1
 Keepout region



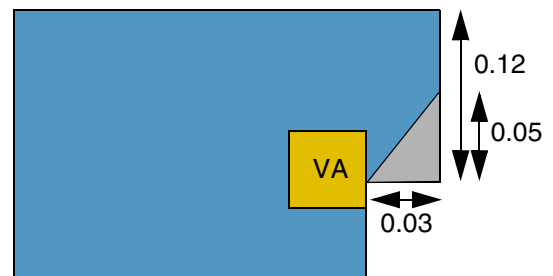
a) PASS. The via cut is outside both keepout regions.



b) FAIL. The via cut overlaps the keepout region.



c) The constraint does not apply because the keepout region touches an end-of-line edge (the shorter edge has width 0.06 (<0.07) and the length of both its adjoining edges is 0.2 (>0.05)).



d) PASS. The via cut is outside the keepout region. (The vertical edge of the keepout region is equal to the constraint value and the horizontal edge is equal to the smaller edge length of 0.03.)

Virtuoso Technology Data Constraints Reference

Extension Constraints

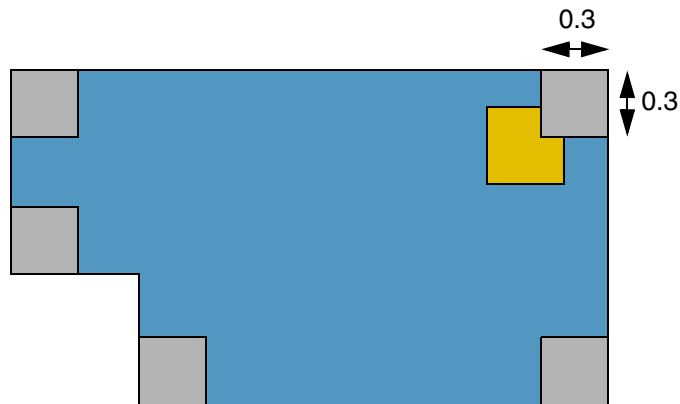
Example 2: minExtensionToCorner with toConvexCorner

The distance of a Via1 via cut from the convex corners of an enclosing Metal1 shape must be at least 0.3.

Note: Only 'toConvexCorner' is specified; therefore the keepout region is a square.

```
orderedSpacings(  
  ( minExtensionToCorner "Metal1" "Via1"  
    'toConvexCorner  
    0.3  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1
■ Keepout region



FAIL. The via cut overlaps the square keepout region.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 3: *minExtensionToCorner* with *toConcaveCorner*, *parallelExtension*, *width*, and *length*

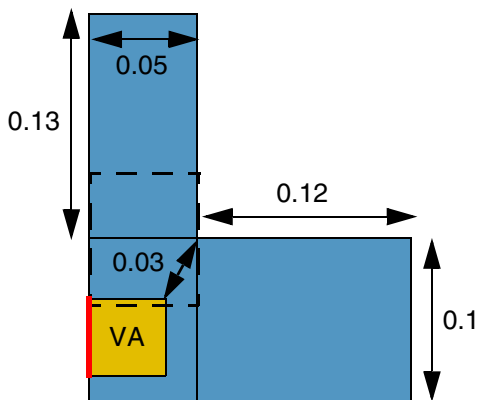
The distance of a VA via cut on layer Via1 from the concave corners of an enclosing Metal1 shape must be at least 0.03. The constraint applies only if the following conditions are true:

- The width of the enclosing Metal1 wire is less than or equal to 0.08.
- The length of the edge that forms a concave corner with the Metal1 wire with width less than or equal to 0.08 is greater than or equal to 0.05.
- The via cut lies inside a search window that is formed by extending the concave corner by 0.03 (the constraint value) on both sides in the direction that is perpendicular to the edge with width less than or equal to 0.08.

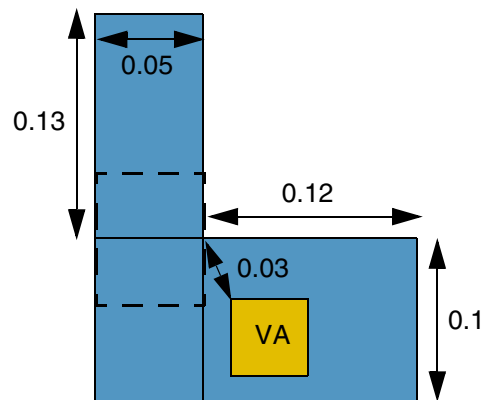
Additionally, the edge of the via cut parallel to the metal edge with width less than or equal to 0.08 has enclosure greater than or equal to 0.02.

```
orderedSpacings (  
  ( minExtensionToCorner "Metal1" "Via1"  
    'toConcaveCorner  
    'parallelExtension 0.02 'width 0.08 'length 0.05  
    'cutClass "VA"  
    0.03  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1
□ Search window



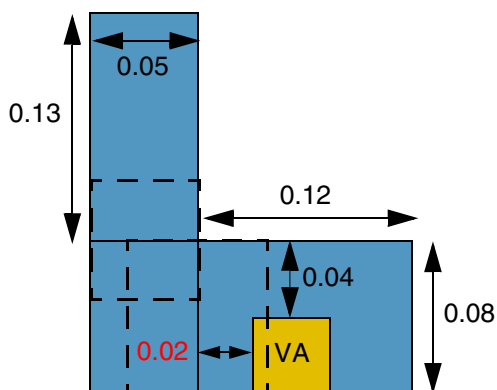
a) FAIL. The vertical wire is 0.05 (<0.08) wide and one edge of the concave corner is 0.12 (>0.05). The via cut is inside the search window, but the enclosure on the left edge of the via cut (in red) is 0 (<0.02).



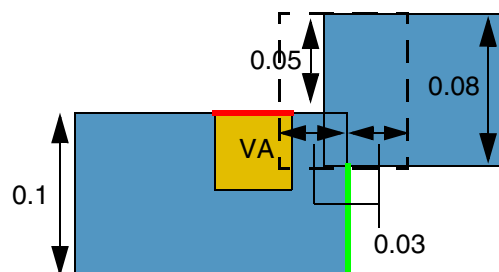
b) The constraint does not apply because the via cut lies outside the search window.

Virtuoso Technology Data Constraints Reference

Extension Constraints



c) FAIL. Both wires fulfill width and length conditions. The via cut is inside the search window, but the enclosure on the bottom edge of the via cut (in red) is 0 (<0.02).



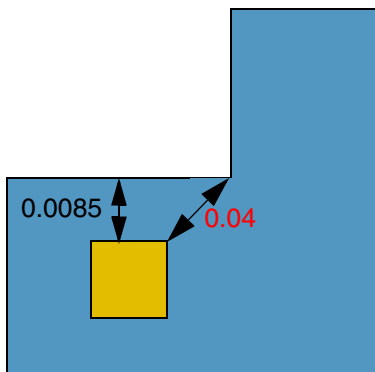
d) FAIL. The bottom concave corner fulfills the width and length conditions (the wire on the right is 0.08 wide and the length of the other concave edge (in green) is 0.07 (>0.05)). Therefore, the search window is formed from the bottom concave corner. However, the enclosure on the top edge of the via cut (in red) is 0 (<0.02). The search window is extended irrespective of the jog.

Example 4: *minExtensionToCorner* with *toConcaveCorner*, *viaEdgeType*, and *exceptCornerTouch*

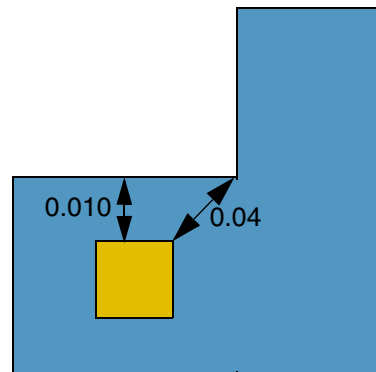
The distance of a Via1 via cut edge of viaEdgeType 1 from a concave corner of an enclosing Metal1 shape must be at least 0.05 (via cut edges with Metal1 extensions less than or equal to 0.009 are of viaEdgeType 1). The constraint does not apply if the via cut touches the concave corner.

```
spacings(
  ( viaEdgeType "Via1"
    'edgeExtension 0.009
    1
  )
); spacings
orderedSpacings(
  ( minExtensionToCorner "Metal1" "Via1"
    'toConcaveCorner
    'viaEdgeType 1
    'exceptCornerTouch
    0.05
  )
); orderedSpacings
```

■ Via1
■ Metal1



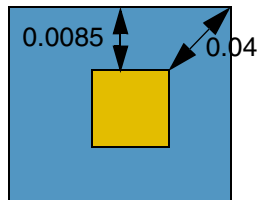
a) FAIL. The extension of the top edge is 0.0085 (<0.009). However, the distance of this edge from the concave corner is 0.04 (<0.05).



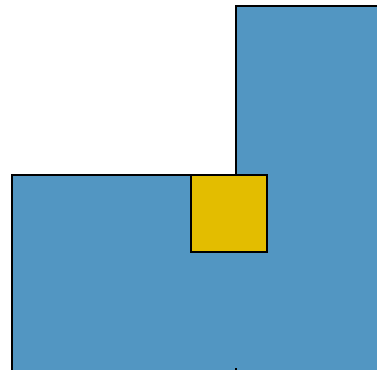
b) The constraint does not apply because the extension of the top edge is 0.01 (>0.009).

Virtuoso Technology Data Constraints Reference

Extension Constraints



c) The constraint does not apply to convex corners.



d) The constraint does not apply because the via cut touches the corner.

Virtuoso Technology Data Constraints Reference

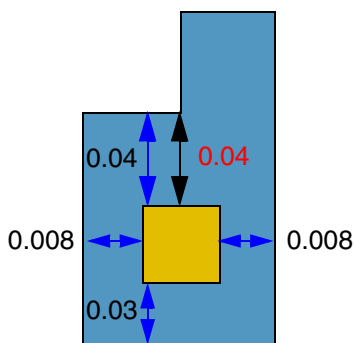
Extension Constraints

Example 5: *minExtensionToCorner* with *toConcaveCorner*, *viaEdgeType*, and *exceptCornerTouch*

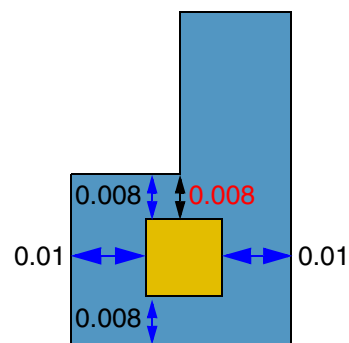
The distance of a Via1 via cut edge of viaEdgeType 1 from a concave corner of an enclosing Metal1 shape must be at least 0.05 (via cut edges with Metal1 extensions greater than 0.009 or a pair of opposite edges with extensions less than or equal to 0.009 are of viaEdgeType 1).

```
spacings(
  ( viaEdgeType "Via1"
    'edgeExtension 0.009
    'negateEdgeExtension
    'anyOppositeExtension 0.009
    1
  )
); spacings
orderedSpacings(
  ( minExtensionToCorner "Metal1" "Via1"
    'toConcaveCorner
    'viaEdgeType 1
    0.05
  )
); orderedSpacings
```

■ Via1
■ Metal1



a) FAIL. The extension of the top edge is 0.04 (>0.009). However, the distance of the top edge from the concave corner is 0.04 (<0.05).



b) The constraint does not apply because the extension of the top edge is 0.008 (less than or equal to 0.009).

Virtuoso Technology Data Constraints Reference

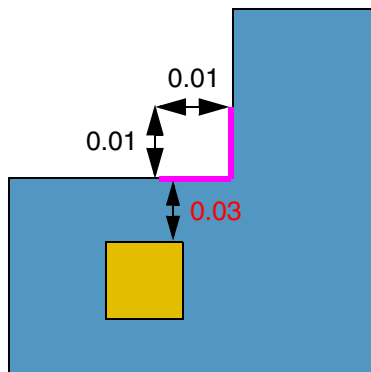
Extension Constraints

Example 6: minExtensionToCorner with toConcaveCorner and extendCornerBy

The distance of a Via1 via cut from a concave corner of an enclosing Metal1 shape must be at least 0.05. The spacing is checked after extending the concave corner by 0.01 in both directions.

```
orderedSpacings (
  ( minExtensionToCorner "Metal1" "Via1"
    'toConcaveCorner
    'extendCornerBy 0.01
    0.05
  )
) ;orderedSpacings
```

■ Via1
■ Metal1

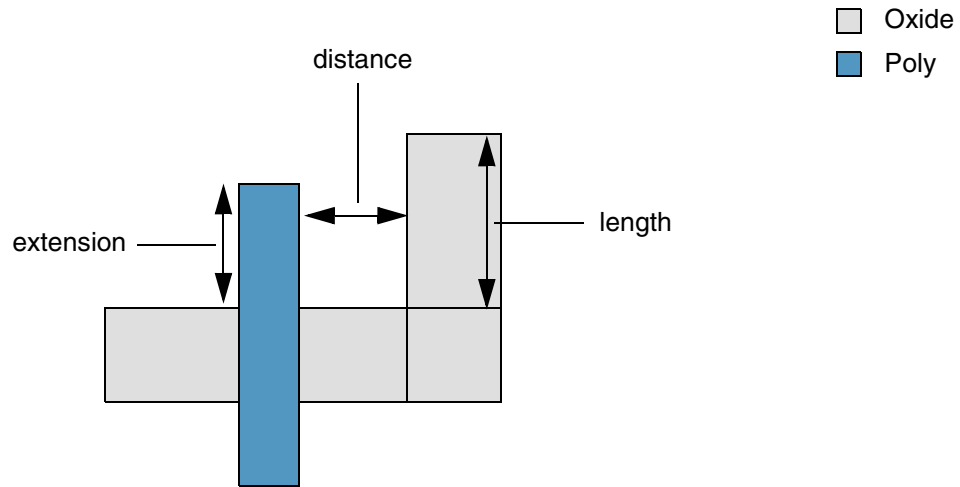


FAIL. The distance of the via cut from the concave corner (after the corner has been extended by 0.1) is 0.03 (< 0.05).

minGateExtension

```
orderedSpacings(  
    ( minGateExtension tx_layer1 tx_layer2  
      'distance f_distance  
      'length f_length  
      f_extension  
    )  
) ;orderedSpacings
```

Specifies the minimum poly *extension* for a gate when an L-shaped oxide shape is at a distance less than or equal to *distance* from the poly shape. The constraint applies only if the length of the jog is greater than *length*.



Values

<i>tx_layer1</i>	The poly layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The oxide layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension of a poly shape past an oxide shape must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

'distance *f_distance*

The constraint applies only if the distance between the poly shape and the L-shaped jog is less than this value.

'length *f_length*

The constraint applies only if the length of the L-shaped jog is greater than this value.

minInnerVertexProximityExtension

```
orderedSpacings (
    ( minInnerVertexProximityExtension tx_layer1 tx_layer2
      ['cutClass {f_width | (f_width f_length) | t_name}]
      'width f_width
      'jogHeight f_jogHeight
      ['withinRange g_range]
      ['otherExtension f_otherExtension]
      f_extension
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum extension of a *layer1* shape past a *layer2* shape that is near an inner vertex of the enclosing *layer1* shape.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension on two opposite edges of the <i>layer2</i> shape—edges that are perpendicular to the edge that faces the inner vertex—must be greater than or equal to this value.

Parameters

<code>'cutClass {f_width (f_width f_length) t_name}</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint).
■ <i>f_width</i> : Width	
■ <i>f_length</i> : Length	
■ <i>t_name</i> : Name of the cut class	

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'width f_width`

The constraint applies only if the width of the *layer1* shape is less than or equal to this value.

`'jogHeight f_jogHeight`

The constraint applies only if the height of the jog is greater than this value.

`'withinRange g_range`

The constraint applies only if the distance of the *layer2* shape from the inner vertex of the metal shape falls in this range.

Note: If this parameter is not specified, the constraint is met only if the *layer2* shape is at a distance greater than or equal to the constraint value (*extension*) from the inner vertex.

Type: Floating-point values specifying a range of distances that trigger the constraint

`'otherExtension f_otherExtension`

The constraint applies only if the extension on the opposite side of the inner vertex is less than this value.

Examples

- Example 1: minInnerVertexProximityExtension with cutClass, width, jogHeight, withinRange, and otherExtension
- Example 2: minInnerVertexProximityExtension with width, jogHeight, and otherExtension

Virtuoso Technology Data Constraints Reference

Extension Constraints

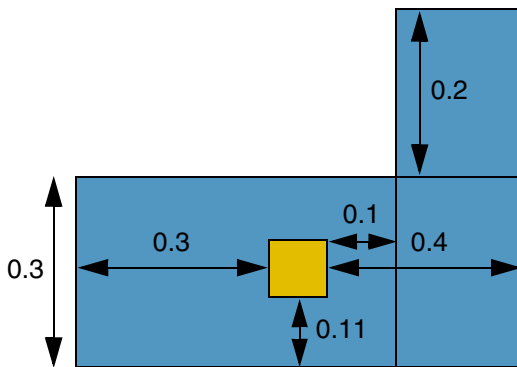
Example 1: minInnerVertexProximityExtension with cutClass, width, jogHeight, withinRange, and otherExtension

The Metal1 extensions on two opposite sides of a Via1 via cut that is at a distance greater than or equal to 0.1 and less than or equal to 0.2 from the inner vertex of the Metal1 shape must be at least 0.35 if the following conditions are met:

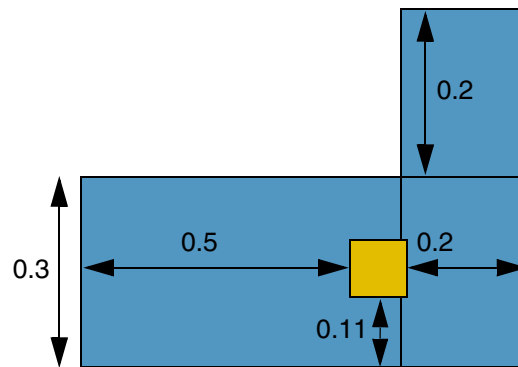
- Width of the enclosing Metal1 shape is less than or equal to 0.4
- Height of the jog is greater than 0.18
- Extension on the cut edge that is opposite to the inner vertex is less than 0.12

```
orderedSpacings (  
  ( minInnerVertexProximityExtension "Metal1" "Via1"  
    'width 0.4  
    'jogHeight 0.18  
    'withinRange "[0.1 0.2]"  
    'otherExtension 0.12  
    0.35  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1



a) FAIL. The width of the Metal1 shape that encloses the via cut is 0.3 (≤ 0.4); the height of the jog is 0.2 (> 0.18); the distance of the via cut from the inner vertex is 0.1 (falls in the specified range); and the extension of the edge that is opposite to the inner vertex is 0.11 (< 0.12). The minimum extension required on the left and right edges of the via cut is 0.35, which is not met on the left side (0.3).



b) The constraint does not apply because the cut to inner vertex spacing (0) does not fall in the specified range ($0.1 \leq \text{spacing} \leq 0.2$).

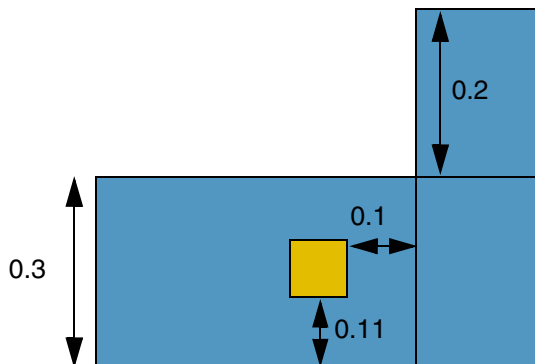
Example 2: minInnerVertexProximityExtension with width, jogHeight, and otherExtension

The 'widthRange parameter is not specified; therefore, the constraint passes only if the distance of the Via1 via cut from the inner vertex of the Metal1 shape is at least 0.15 when the following conditions are met:

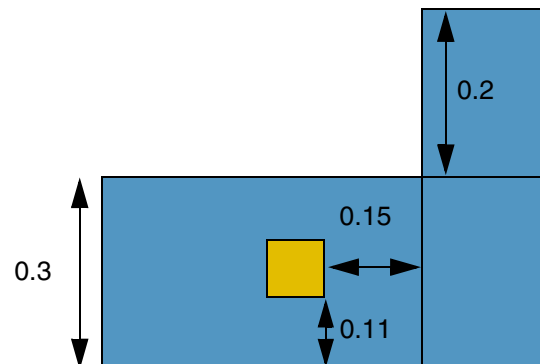
- Width of the enclosing Metal1 shape is less than or equal to 0.4
- Height of the jog is greater than 0.18
- Extension on the cut edge that is opposite to the inner vertex is less than 0.12

```
orderedSpacings (
  ( minInnerVertexProximityExtension "Metal1" "Via1"
    'width 0.4
    'jogHeight 0.18
    'otherExtension 0.12
    0.15
  )
) ;orderedSpacings
```

■ Via1
■ Metal1



a) FAIL. The width of the Metal1 shape that encloses the via cut is 0.3 (≤ 0.4); the height of the jog is 0.2 (> 0.18); and the extension of the edge that is opposite to the inner vertex is 0.11 (< 0.12). However, the distance of the via cut to the inner vertex is only 0.1 (< 0.15).

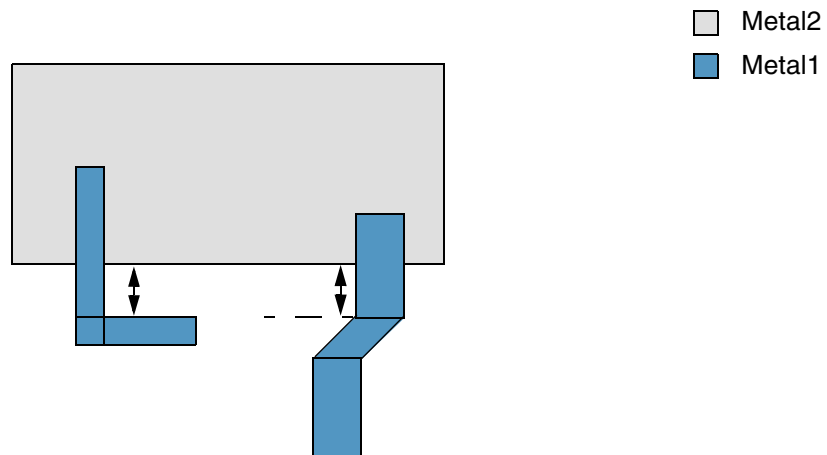


b) PASS. The distance of the via cut to the inner vertex must be 0.15, a condition that is met.

minInsideCornerExtension

```
orderedSpacings (
    ( minInsideCornerExtension tx_layer1 tx_layer2
      f_extension
    )
) ;orderedSpacings
```

Specifies the distance between an inside corner of a *layer1* shape and a *layer2* edge that faces the inside corner. The constraint applies to an inside corner formed by a *layer1* shape at any angle.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_extension</i>	The distance between an inside corner of the <i>layer1</i> shape and the <i>layer2</i> edge that faces the inside corner must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

None

Example

The distance of a concave corner of a Metal1 shape from the edge of a Metal2 shape that faces the concave corner must be 1.0.

```
orderedSpacings (
  ( minInsideCornerExtension "Metal1" "Metal2"
    1.0
  )
) ;orderedSpacings
```

minNeighborExtension

```
orderedSpacings (
  ( minNeighborExtension tx_metalLayer tx_cutLayer
    'cutClass {f_width | (f_width f_length) | t_name}
    'paraLengthRange g_paraLengthRange
    'withinRange g_withinRange
    ['width f_wireWidth]
    ['otherExtension f_otherExtension
      ['layer tx_otherMetalLayer]
      ['dualExtension f_ext1 f_ext2]
    ]
    ['mask1 | 'mask2 | 'mask3 | 'mask4]
    (f_ext f_oppExt)
  )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum extension of a metal shape past a cut shape if the distance between the cut shape and a neighboring wire falls in *withinRange* and the parallel run length between them falls in *paraLengthRange*. You can specify both a lower and an upper bound for *withinRange* and *paraLengthRange*. Only one such neighbor must exist when only the lower bound for *withinRange* and *paraLengthRange* is specified.

Additionally, for the constraint to apply, the neighboring wire must have an edge in the preferred routing direction facing the cut shape and this neighboring wire edge must not have a parallel run length greater than or equal to zero with the cut shape edge orthogonal to it.

The constraint specifies a pair of extension values—of which one applies to one set of opposite sides/edges of the cut shape and the other applies to the other set of opposite sides/edges of the cut shape. For example, if the first value is used to evaluate the extensions in the horizontal direction, the second value is used to evaluate the extensions in the vertical direction.

Optionally, the constraint applies only if the width of the metal shape (the shape drawn on *metalLayer*) enclosing the cut shape is less than *wireWidth*.

Values

<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Extension Constraints

tx_cutLayer

The cut layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_ext f_oppExt

The extension values.

- *f_ext*: The minimum extension on opposite sides in one direction, horizontal or vertical.
- *f_oppExt*: The minimum extension on opposite sides in the perpendicular direction.

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'paraLengthRange g_paraLengthRange`

The lower and, optionally, the upper bound for the parallel run length between the cut shape and the neighboring wire.

Note: If upper bound is defined, it must be defined for both `'paraLengthRange` and `'withinRange`.

When only the lower bound is specified: The constraint applies if the parallel run length between the cut and the neighboring wire is greater than or equal to the lower bound.

When both lower and upper bounds are specified:

- The constraint does not apply if neighboring wires are present on both sides of the cut shape inside the small search window defined using the lower bound values.
- The constraint applies if a neighboring wire is present on one or both sides of the cut shape at a distance greater than or equal to the lower bound and less than the upper bound with parallel run length less than or equal to the lower bound and greater than the upper bound (that is, the neighboring wires must be inside the outer search window defined using the upper bound values).

Note: The upper bound for `'withinRange` must be greater than the lower bound. The upper bound for `'paraLengthRange` must be less than the lower bound.

Type: Floating-point values specifying a range of parallel run lengths that trigger the constraint

`'withinRange g_withinRange`

The lower and, optionally, the upper bound for the distance between the cut and the neighboring wire.

Note: If upper bound is defined, it must be defined for both `'paraLengthRange` and `'withinRange`.

When only the lower bound is specified: The constraint applies if the distance between the cut shape and the neighboring wire is less than the lower bound.

When both lower and upper bounds are specified:

- The constraint does not apply if neighboring wires are present on both sides of the cut shape inside the small search window defined using the lower bound values.
- The constraint applies if a neighboring wire is present on one or both sides of the cut shape at a distance greater than or equal to the lower bound and less than the upper bound with parallel run length less than or equal to the lower bound and greater than the upper bound (that is, the neighboring wires must be inside the outer search window defined using the upper bound values).

Note: The upper bound for `'withinRange` must be greater than the lower bound. The upper bound for `'paraLengthRange` must be less than the lower bound.

Type: Floating-point values specifying a range of distances that trigger the constraint

`'width f_wireWidth`

The constraint applies only if the width of the metal shape (the shape drawn on *metalLayer*) enclosing the cut shape is less than this value.

`'otherExtension f_otherExtension`

The constraint applies only if the extension of the "below" metal shape is less than this value on either side of the cut shape in a direction perpendicular to the direction of the "above" metal shape containing the cut shape.

You can specify a metal layer other than the default "below" metal layer by using the `'layer` parameter.

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'layer tx_otherMetalLayer`

The layer on which the `'otherExtension` parameter applies.

Type: String (layer name) or Integer (layer number)

`'dualExtension f_ext1 f_ext2`

The constraint does not apply if the extension of *otherMetalLayer* shape past the cut shape:

- Is greater than or equal to the smaller of the two specified extension values on all four sides, including the corners
- Is greater than or equal to the larger of the two specified extension values on any two opposite sides of the cut shape

`'mask1 | 'mask2 | 'mask3 | 'mask4`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the cut shape is enclosed by a shape on the specified mask on the metal layer on which the constraint is specified.

Type: Boolean

Examples

- Example 1: minNeighborExtension with cutClass, paraLengthRange, withinRange, and width
- Example 2: minNeighborExtension with cutClass, paraLengthRange, and withinRange
- Example 3: minNeighborExtension with cutClass, paraLengthRange, withinRange, otherExtension, and layer
- Example 4: minNeighborExtension with cutClass, paraLengthRange, withinRange, otherExtension, layer, and mask1

Virtuoso Technology Data Constraints Reference

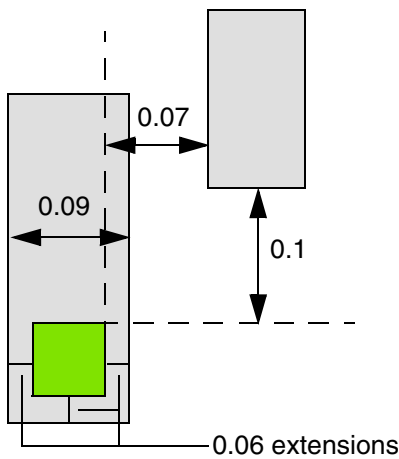
Extension Constraints

Example 1: *minNeighborExtension* with *cutClass*, *paraLengthRange*, *withinRange*, and *width*

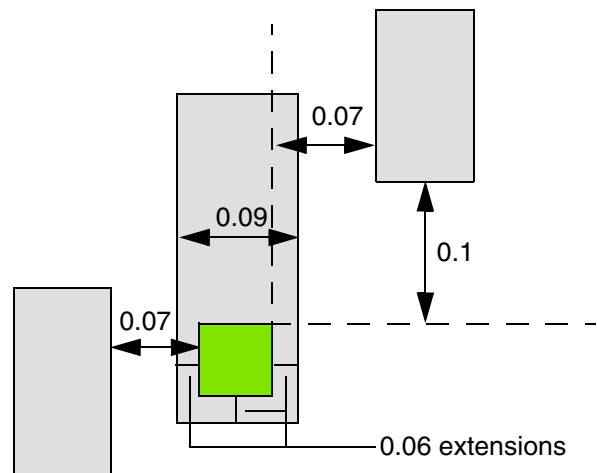
The extensions of a Metal2 shape on all sides of a Via2 cut shape with dimensions 0.03x0.03 must be at least 0.09 if a neighboring wire is present at a distance less than 0.08 from the cut shape and has parallel run length greater than or equal to -0.11 with the cut shape.

```
spacingTables (
  ( minOppExtension "Metal2" "Via2"
    ( ( "width" nil nil )
      'cutClass 0.03
    )
    (
      0.0 ((0.05 0.0))
      0.1 ((0.06 0.0))
    )
  )
) ;spacingTables
orderedSpacings (
  ( minNeighborExtension "Metal2" "Via2"
    'cutClass 0.03
    'paraLengthRange -0.11
    'withinRange 0.08
    'width 0.1
    (0.09 0.09)
  )
) ;orderedSpacings
```

■ Via2
■ Metal2



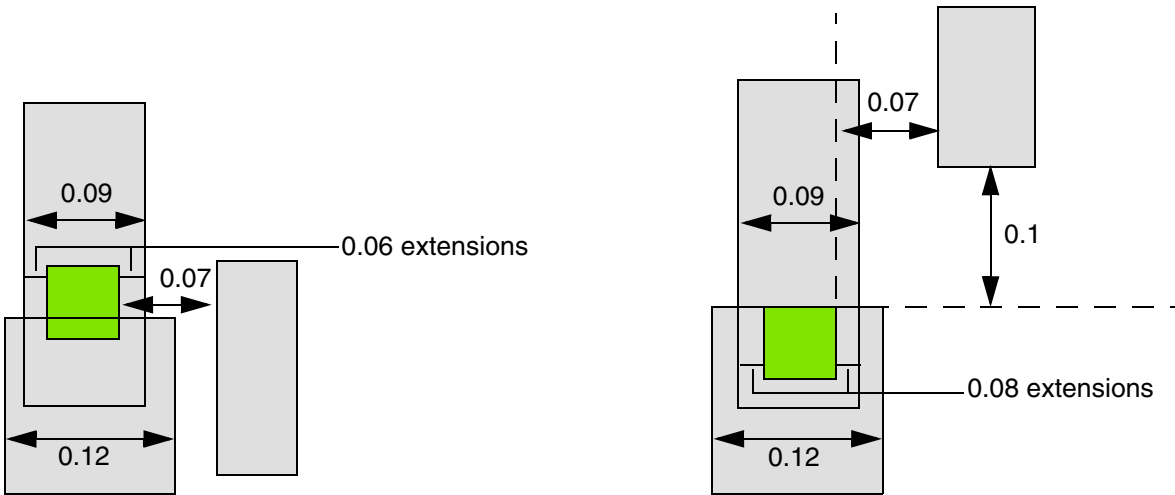
a) **FAIL.** The distance of the neighboring wire from the via cut is 0.07 (<0.08) and their parallel run length is -0.1 (>-0.11). However, Metal2 extensions on three sides are 0.06 (<0.09).



b) **PASS.** The *minNeighborExtension* constraint does not apply because there are two neighboring wires at a distance less than 0.08 from the via cut, instead of just one wire. The *minOppExtension* constraint applies and the extension requirement of (0.05 0.00) is met.

Virtuoso Technology Data Constraints Reference

Extension Constraints



c) FAIL. The via cut is not fully enclosed by the wide wire. As a result, the extension requirement of 0.09 is not met on the left and right sides of the via cut.

d) PASS. The `minNeighborExtension` constraint does not apply because the width of the wire containing the via cut is 0.12 (>0.1). The `minOppExtension` constraint applies and the extension requirement of (0.06 0.00) is met.

Example 2: `minNeighborExtension` with `cutClass`, `paraLengthRange`, and `withinRange`

The extensions of a Metal2 shape on all sides of a Via2 cut shape with dimensions 0.03x0.03 must be at least 0.015 if a neighboring wire is present:

- On one side of the cut shape at a distance less than 0.04 from the cut shape and has parallel run length greater than or equal to -0.08 with the cut shape, that is, inside the small search window.
- On one or both sides of the cut shape at a distance greater than or equal to the lower bound (0.04) and less than the upper bound (0.06), with parallel run length less than or equal to the lower bound (-0.08) and greater than the upper bound (-0.1), that is, inside the outer search window.

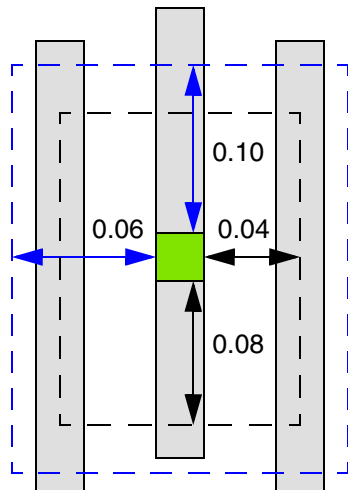
Virtuoso Technology Data Constraints Reference

Extension Constraints

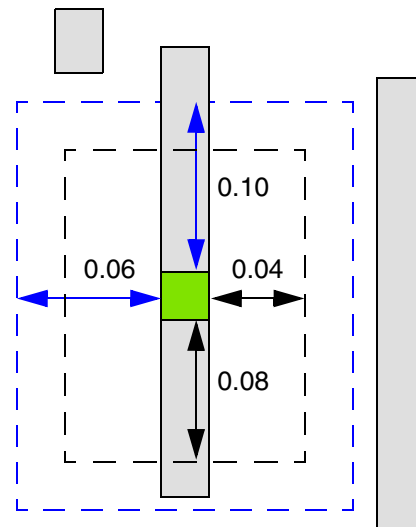
Because upper bounds are defined for parallel run length and distance (-0.1 and 0.06, respectively), the constraint does not apply if a neighboring wire is present on both sides of the cut shape inside the small search window.

```
orderedSpacings (
  ( minOppExtension "Metal2" "Via2"
    'cutClass 0.03
    (0.02 0.0)
  )
) ;orderedSpacings
orderedSpacings (
  ( minNeighborExtension "Metal2" "Via2"
    'cutClass 0.03
    'paraLengthRange "[-0.08 -0.1]"
    'withinRange "[0.04 0.06]"
    (0.015 0.015)
  )
) ;orderedSpacings
```

■ Via2
■ Metal2



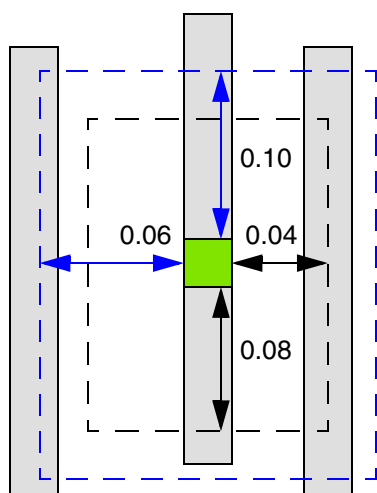
a) PASS. The `minNeighborExtension` constraint does not apply because neighboring wires are present inside the smaller search window on both sides of the via cut. The `minOppExtension` constraint applies and the extension requirement of (0.02 0.0) is met.



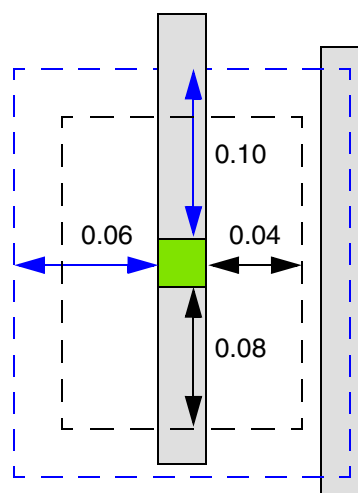
b) PASS. The `minNeighborExtension` constraint does not apply because a neighboring wire is not present inside either search window. The `minOppExtension` constraint applies and the extension requirement of (0.02 0.0) is met.

Virtuoso Technology Data Constraints Reference

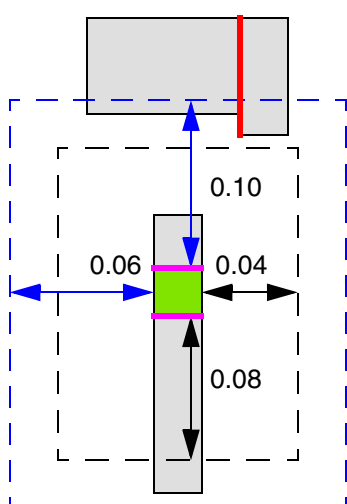
Extension Constraints



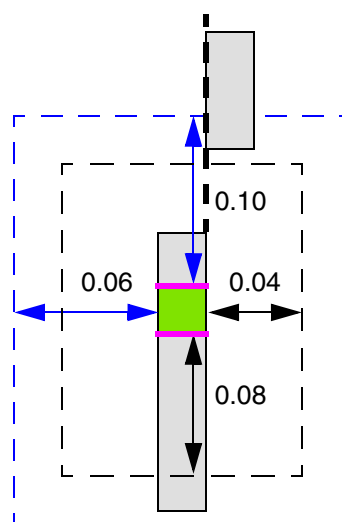
c) FAIL. A neighboring wire is present inside the small search window. However, the via cut does not meet the extension requirement of 0.015 on all four sides.



d) FAIL. A neighboring wire is present inside the outer search window (that is, at a distance greater than or equal to 0.04 and less than 0.06). However, the via cut does not meet the extension requirement of 0.015 on all four sides.



e) FAIL. The constraint applies because the edge of the neighboring wire in the preferred routing direction (shown in red) has less than zero parallel run length with the via cut edge orthogonal to it (shown in pink). However, the via cut does not meet the extension requirement of 0.015 on all four sides.



f) The constraint does not apply because the edge of the neighboring wire in the preferred routing direction has parallel run length equal to zero with the via cut edge orthogonal to it (shown in pink).

Example 3: minNeighborExtension with cutClass, paraLengthRange, withinRange, otherExtension, and layer

The extensions of a Metal2 shape on all sides of a Via2 cut shape with dimensions 0.03x0.03 must be at least 0.015 if all of the following conditions are met:

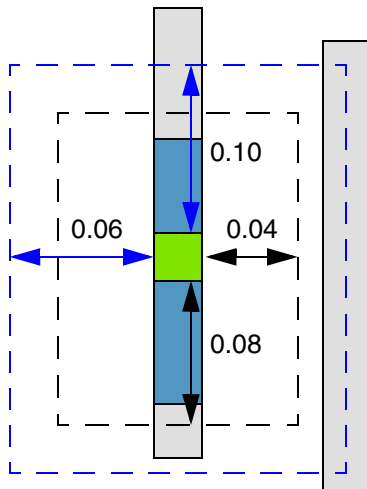
- The extension of the "below" metal (Metal1) shape is less than 0.05 on either side of the cut shape in a direction perpendicular to the direction of the "above" metal (Metal2) shape containing the cut shape.
- A neighboring wire is present:
 - On one side of the cut shape at a distance less than 0.04 from the cut shape and has parallel run length greater than or equal to -0.08 with the cut shape, that is, inside the small search window.
 - On one or both sides of the cut shape at a distance greater than or equal to the lower bound (0.04) and less than the upper bound (0.06), with parallel run length less than or equal to the lower bound (-0.08) and greater than the upper bound (-0.1), that is, inside the outer search window.

Virtuoso Technology Data Constraints Reference

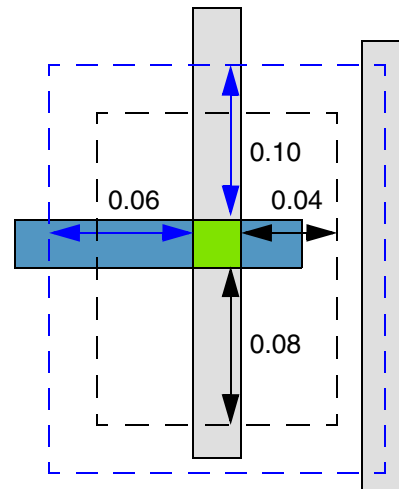
Extension Constraints

```
orderedSpacings (
  ( minOppExtension "Metal2" "Via2"
    'cutClass 0.03
    (0.02 0.0)
  )
) ;orderedSpacings
orderedSpacings (
  ( minNeighborExtension "Metal2" "Via2"
    'cutClass 0.03
    'paraLengthRange "[-0.08 -0.1]"
    'withinRange "[0.04 0.06]"
    'otherExtension 0.05
    'layer "Metal1"
    (0.015 0.015)
  )
) ;orderedSpacings
```

■ Via2
■ Metal2
■ Metal1



a) FAIL. The "below" metal, Metal1, extension is 0.0 (< 0.05) on both sides, in the direction perpendicular to the direction of the "above" metal, Metal2, wire containing the cut shape, and a Metal2 wire is present inside the large search window. However, the via cut does not meet the extension requirement of 0.015 on all four sides.



b) FAIL. The Metal1 extension is less than 0.04 (or <0.05) on the right side, in the direction perpendicular to the direction of the Metal2 wire, and a Metal2 wire is present inside the large search window. However, the via cut does not meet the extension requirement of 0.015 on all four sides.

Example 4: minNeighborExtension with cutClass, paraLengthRange, withinRange, otherExtension, layer, and mask1

The extensions of a Metal2 shape on all sides of a Via2 cut shape with dimensions 0.03x0.03 must be at least 0.015 if all of the following conditions are met:

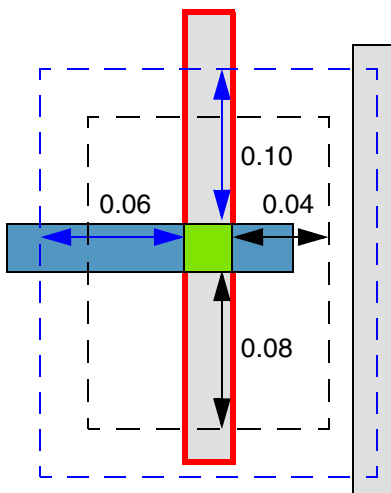
- The extension of the "below" metal (Metal1) shape is less than 0.05 on either side of the cut shape in a direction perpendicular to the direction of the "above" metal (Metal2) shape containing the cut shape.
- A neighboring wire is present:
 - On one side of the cut shape at a distance less than 0.04 from the cut shape and has parallel run length greater than or equal to -0.08 with the cut shape, that is, inside the small search window.
 - On one or both sides of the cut shape at a distance greater than or equal to the lower bound (0.04) and less than the upper bound (0.06), with parallel run length less than or equal to the lower bound (-0.08) and greater than the upper bound (-0.1), that is, inside the outer search window.

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
orderedSpacings(  
  ( minNeighborExtension "Metal2" "Via2"  
    'cutClass 0.03  
    'paraLengthRange "[-0.08 -0.1]"  
    'withinRange "[0.04 0.06]"  
    'otherExtension 0.05  
    'layer "Metal1"  
    'mask1  
    (0.015 0.015)  
  )  
) ;orderedSpacings
```

■ Via2
■ Metal2
■ Metal1
■ mask1



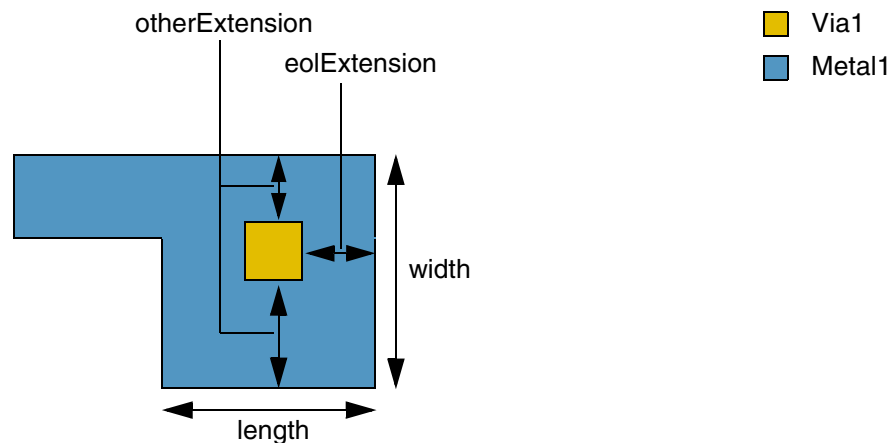
FAIL. The constraint applies because the via cut is enclosed by a Metal2-mask1 wire. Moreover, the Metal1 extension is less than 0.04 (or <0.05) on the right side, in the direction perpendicular to the direction of the Metal2 wire, and a Metal2 wire is present inside the large search window. However, the via cut does not meet the extension requirement of 0.015 on all four sides.

minOppEndOfLineExtension

```
orderedSpacings (
  ( minOppEndOfLineExtension tx_layer1 tx_layer2
    'width f_width ['length f_length] ['eolOnly]
    ['cutClass {f_width | (f_width f_length) | t_name} ['shortEdgeOnEol]]
    ['endExclusion (f_endPerpendicularExcl f_endExcl)
      |
      'sideExclusion (f_sideEdgeExcl f_backExcl f_fwdExcl)
    ]
    ['vertical | 'horizontal]
    (f_eolExtension f_otherExtension)
  )
) ;orderedSpacings
```

Specifies the minimum extension of a *layer1* shape past a *layer2* shape. The constraint applies only if the width of the end-of-line edge is less than *width* and the length of both edges adjoining the end-of-line edge is greater than or equal to *length*.

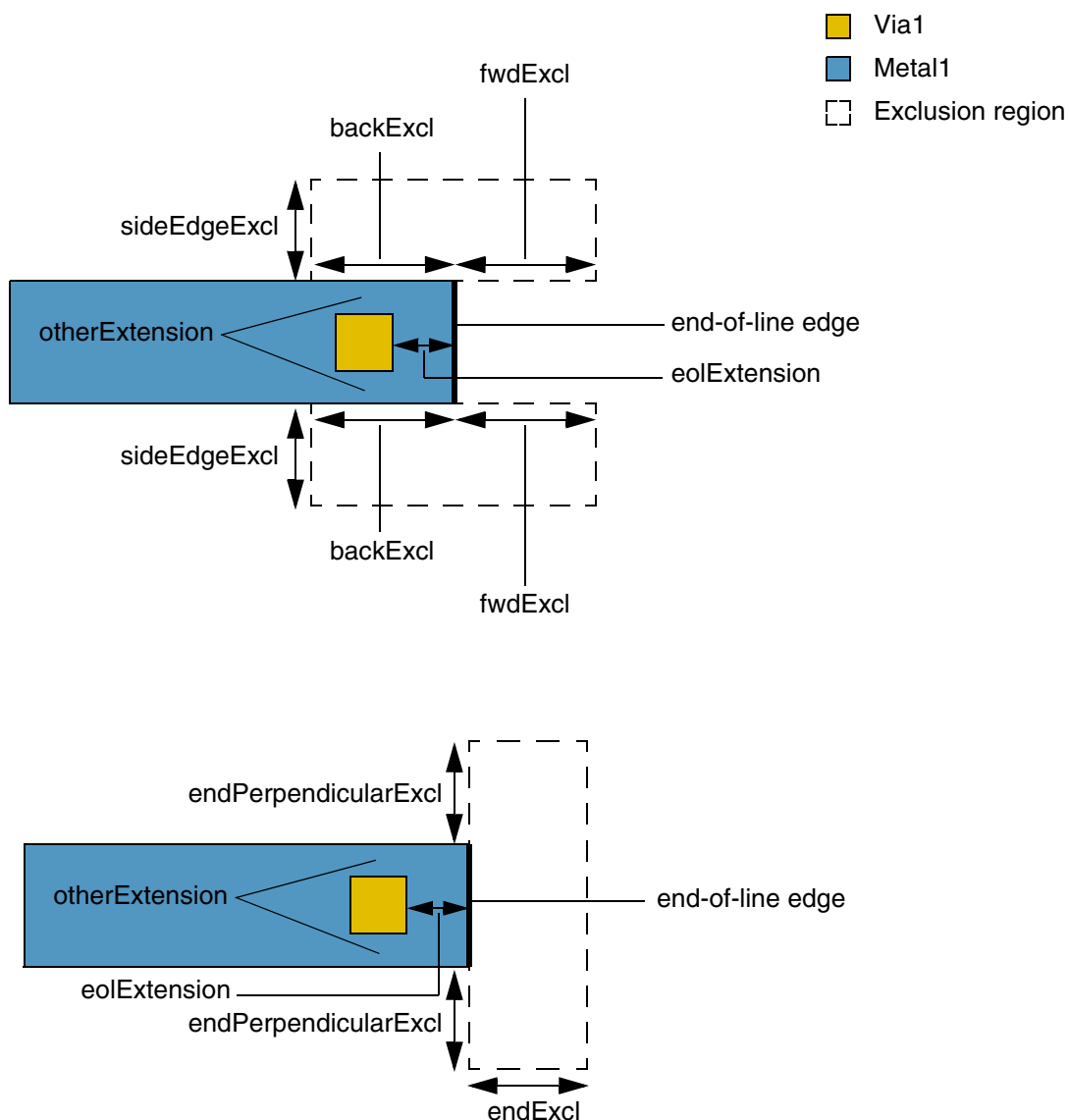
The first extension value applies to the end-of-line edge and the edge opposite it; the second extension value applies to the other pair of edges, as shown below:



Virtuoso Technology Data Constraints Reference

Extension Constraints

Optionally, you can specify an exclusion region as shown below. The constraint does not apply if a neighboring shape on another net overlaps or touches a boundary of this exclusion region.



Virtuoso Technology Data Constraints Reference

Extension Constraints

Values

<i>tx_layer1</i>	The first layer (metal) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer (cut) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_eolExtension</i> <i>f_otherExtension</i>	The extension of the end-of-line edge and the edge opposite to it must be greater than or equal to <i>eolExtension</i> and the extension of the other pair of edges must be greater than or equal to <i>otherExtension</i> .

Parameters

'width <i>f_width</i>	This constraint applies only if the width of the end-of-line edge is less than this value.
'length <i>f_length</i>	This constraint applies only if the length of both edges adjoining an end-of-line edge with width less than <i>width</i> is greater than or equal to this value.
'eolOnly	Only <i>minOppEndOfLineExtension</i> constraints must be satisfied for all end-of-line edges. Other enclosure type constraints specified for the layer pair or cut class, such as <i>minOppExtension</i> , can be ignored. If specified for one <i>minOppEndOfLineExtension</i> constraint, it must be specified for all <i>minOppEndOfLineExtension</i> constraints that are defined for that layer pair or cut class, and, for an end-of-line edge, only one such constraint definition needs to be fulfilled, while other constraint definitions without this parameter are ignored.
'shortEdgeOnEol	The short edges of a rectangular cut shape must have extension greater than or equal to <i>eolExtension</i> and the long edges of the cut shape must have extension greater than or equal to <i>otherExtension</i> . The constraint does not apply if the short edge is not on the end-of-line.

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
'cutClass {f_width | (f_width f_length) | t_name}
```

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

```
'endExclusion (f_endPerpendicularExcl f_endExcl)
```

The two values together define an exclusion region adjoining the end-of-line edge. The constraint does not apply if a neighboring shape overlaps or touches a boundary of this exclusion region.

Note: If both 'endExclusion and 'sideExclusion are specified, the constraint does not apply if a neighboring shape is present in the side exclusion area and the end exclusion area.

```
'sideExclusion (f_sideEdgeExcl f_backExcl f_fwdExcl)
```

The three values together define an exclusion region along the length of the end-of-line shape. The constraint does not apply if a neighboring shape overlaps or touches a boundary of this exclusion region.

Virtuoso Technology Data Constraints Reference

Extension Constraints

'horizontal' | 'vertical'

(Virtuoso Advanced Node for Layout Only) The parameter applies only if the line-end is in the given direction.

Examples

- Example 1: minOppEndOfLineExtension with width
- Example 2: minOppEndOfLineExtension with width and shortEdgeOnEol
- Example 3: minOppEndOfLineExtension with width, endExclusion, and sideExclusion

Virtuoso Technology Data Constraints Reference

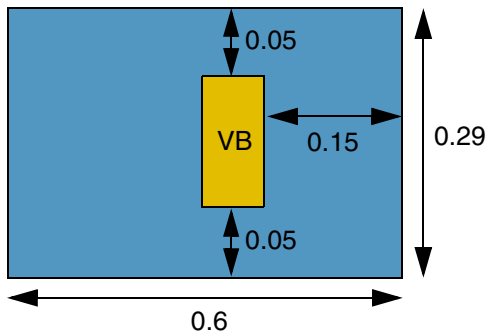
Extension Constraints

Example 1: *minOppEndOfLineExtension with width*

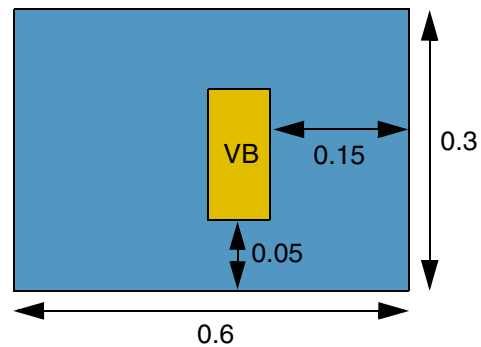
A VB via cut on layer Via1 must have an extension of at least 0.15 on an end-of-line edge and the opposite edge when the width of the enclosing Metal1 shape is less than 0.3; the extensions on the other two sides of the via cut must be at least 0.05.

```
orderedSpacings (  
  ( minOppEndOfLineExtension "Metal1" "Via1"  
    'width 0.3  
    'cutClass "VB"  
    (0.15 0.05)  
  )  
) ;orderedSpacings
```

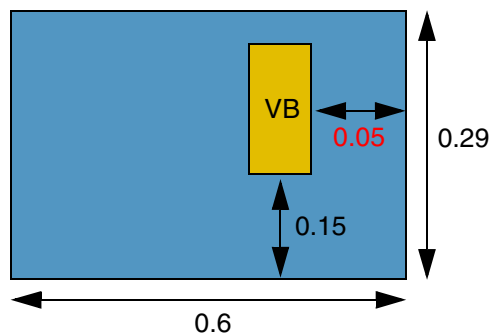
■ Via1
■ Metal1



a) PASS. The width of the end-of-line edge is 0.29 (< 0.3). The Metal1 extensions on the end-of-line edge and the side opposite to it are 0.15 and the extensions on the other two sides of the via cut are 0.05.



b) The constraint does not apply because all Metal1 edges are ≥ 0.30 . As a result, there are no end-of-line edges.



c) FAIL. The width of the end-of-line edge is 0.29 (< 0.3). However, the Metal1 extension on the end-of-line edge is only 0.05 (< 0.15).

Virtuoso Technology Data Constraints Reference

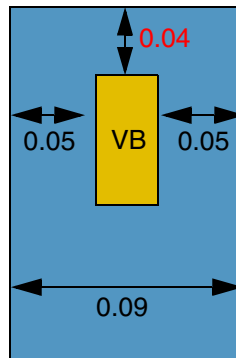
Extension Constraints

Example 2: minOppEndOfLineExtension with width and shortEdgeOnEol

The extensions on the short edges of a VB via cut on layer Via1 must be at least 0.05 if the end-of-line width of the enclosing Metal1 shape is less than 0.1; the extension on the long edges of the via cut must be at least 0.05.

```
orderedSpacings (  
  ( minOppEndOfLineExtension "Metal1" "Via1"  
    'width 0.1 'shortEdgeOnEol 'vertical  
    'cutClass "VB"  
    (0.05 0.04)  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1



FAIL. The width of the end-of-line edge is 0.09 (< 0.1). However, the top short edge of the rectangular via cut has extension less than 0.05.

Virtuoso Technology Data Constraints Reference

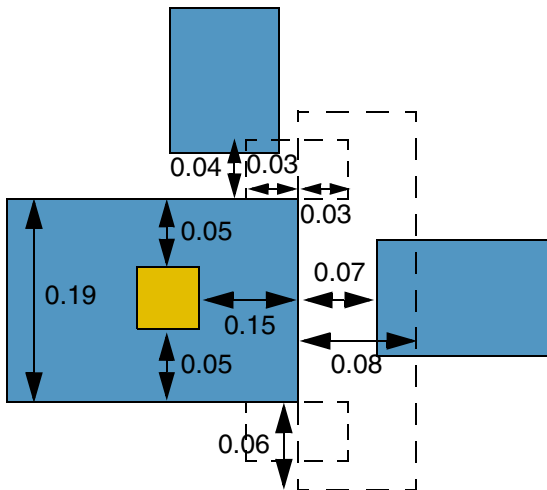
Extension Constraints

Example 3: minOppEndOfLineExtension with width, endExclusion, and sideExclusion

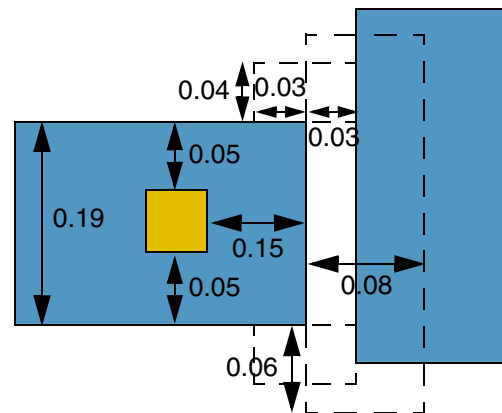
A via cut on layer Via1 must have extensions of at least 0.15 on an end-of-line edge and the opposite edge when the width of the enclosing Metal1 shape is less than 0.2; the extensions on the other two sides of the via cut must be at least 0.05. If a neighboring wire is present within 0.04, with forward and backward extensions of 0.03, of the side edges *and* within 0.08 with extension 0.06 of the end-of-line edge, the via cut must have 0.1 extensions on all edges.

```
orderedSpacings (
  ( minOppEndOfLineExtension "Metal1" "Via1"
    'width 0.2
    (0.1 0.1)
  )
  ( minOppEndOfLineExtension "Metal1" "Via1"
    'width 0.2
    'endExclusion (0.06 0.08)
    (0.15 0.05)
  )
  ( minOppEndOfLineExtension "Metal1" "Via1"
    'width 0.2
    'sideExclusion (0.04 0.03 0.03)
    (0.15 0.05)
  )
) ;orderedSpacings
```

Via1
 Metal1
 Exclusion region



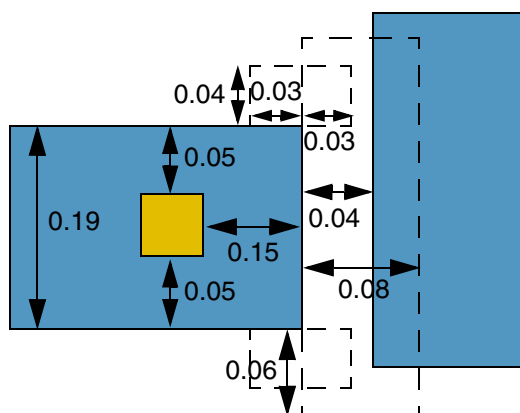
a) FAIL. The width of the end-of-line edge is 0.19 (<0.2). However, the extension requirement of (0.15, 0.05) does not apply because a neighboring wire is present inside both the end exclusion region and the side exclusion region. The extension requirement of (0.1, 0.1) applies, but is not met.



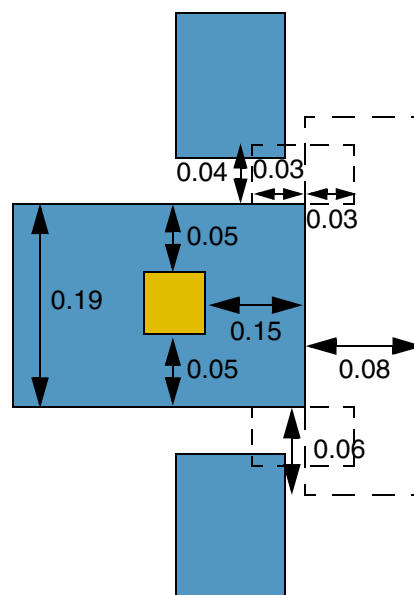
b) FAIL. The width of the end-of-line edge is 0.19 (<0.2). However, the extension requirement of (0.15, 0.05) does not apply because the neighboring wire is inside the end exclusion region and also touches the side exclusion region boundary. The extension requirement of (0.1, 0.1) applies, but is not met.

Virtuoso Technology Data Constraints Reference

Extension Constraints



c) PASS. The width of the end-of-line edge is 0.19 (<0.2) and a neighboring wire is not present inside the side exclusion region. Therefore, the extension requirement of (0.15, 0.05) applies and is met.



d) PASS. The width of the end-of-line edge is 0.19 (<0.2) and a neighboring wire is not present inside the end exclusion region. Therefore, the extension requirement of (0.15, 0.05) applies and is met.

Virtuoso Technology Data Constraints Reference

Extension Constraints

minOppExtension

```
orderedSpacings(  
  ( minOppExtension tx_layer1 tx_layer2  
    ['cutClass {f_width | (f_width f_length) | t_name} ['endSide]]  
    ['widthHorizontal | 'widthVertical]  
    ['cutDistance f_cutDistance ['noSharedEdge | 'hasParaRunLength]]  
    ['length f_length  
      | 'extraCut ['extraOnly]  
      | 'redundantCutWithin f_distance  
    ]  
    ['stepSizePair (f_val1 f_val2)]  
    ['area f_area]  
    ['horizontal | 'vertical]  
    ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)  
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]  
      |  
      ['outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)  
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]  
      ]  
    ['distanceWithin f_distance 'nearbyLayer tx_layer]  
    ['otherLayer tx_otherLayer  
      'otherWidth f_otherWidth  
      {'withinHorizontal | 'withinVertical}  
      'parallelWithin f_parallelWithin  
      ['otherExtension f_otherExtension]  
    ]  
    ['alignedLayer g_layer 'alignedHorizontal | 'alignedVertical]  
    (f_ext f_oppExt) ['coincidentAllowed]  
  )  
);orderedSpacings  
  
spacingTables(  
  ( minOppExtension tx_layer1 tx_layer2  
    (( "width" nil nil )  
      ['widthHorizontal | 'widthVertical]  
      ['cutClass {f_width | (f_width f_length) | t_name}  
        ['endSide ['offCenterline]]  
      ]  
      ['cutDistance f_cutDistance  
        ['noSharedEdge | 'hasParaRunLength | 'hasExactPrl f_hasExactPrl]  
      ]  
      ['length f_length  
        | 'extraCut ['extraOnly ['exactPrl f_exactPrl]]  
        | 'redundantCutWithin f_distance  
      ]  
      ['exceptCutMetalEdgeExt | 'useMinWidth]  
      ['stepSizePair (f_val1 f_val2)]  
      ['horizontal | 'vertical]  
      ['includeAbutted] ['minCorner]  
      ['jogLengthOnly f_jogLength ['treatLAsJog]  
    )  
  )  
);spacingTables
```

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
        {'horizontalJog | 'verticalJog}
        'jogWireWidth f_jogWireWidth
    ]
    ['hollowHorizontal f_horiLength | 'hollowVertical f_vertLength
    ['hollowOffset f_offset]]
    ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['distanceWithin f_distance 'nearbyLayer tx_layer]
    [f_default]
    ['alignedLayer g_layer 'alignedHorizontal | 'alignedVertical]
    )
    (g_table) ['coincidentAllowed]
)
) ;spacingTables
```

Specifies the minimum extension of a shape on *layer1* past a shape on *layer2*. Typically, *layer1* is a metal layer and *layer2* is a cut layer.

The constraint specifies a pair of extension values, of which one applies to one set of opposite sides/edges of the *layer2* shape and the other applies to the other set of opposite sides/edges of the *layer2* shape. For example, if the first value is used to evaluate the extensions in the horizontal direction, the second value is used to evaluate the extensions in the vertical direction. Moreover, the extension values can apply in a different direction for different shapes on the specified layer.

Optionally, the `minOppExtension` constraint can be set to apply only in a certain direction. In this case, the extensions in the specified direction must satisfy the first constraint value, and the extensions in the perpendicular direction must satisfy the second constraint value. The constraint can also be used to specify multiple pairs of extension values that are applied based on the width of the *layer1* shape.

In some processes, the constraint can be applied only to rectangular cut shapes that do not collide with the centerline of the wire. Additionally, for rectangular cut classes, the first extension value can be applied to the short edges and the second extension value can be applied to the long edges.

This constraint definition is not symmetric, which means that the minimum extension of a shape on *layer1* past a shape on *layer2* is not the same as the minimum extension of a shape on *layer2* past a shape on *layer1*. Therefore, if you specify the `minOppExtension` constraint for a layer pair, you should not specify the `minExtension` constraint for that layer pair.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Multiple `minOppExtension` constraints can be defined in a precedence constraint group. In such a constraint group, the constraints are ordered first by width and cut distance, then by width, and finally by length. Members in the group that do not specify width have the lowest precedence.

An OR constraint group should contain settings for only the same constraint, with different parameters. If any of the settings is satisfied, the constraint requirement is met. An exception is made for `minOppExtension`, `minQuadrupleExtension`, and `minExtensionDistance` constraints, which can be bound together in an OR group, and if any of these constraints is satisfied, the extension requirement is met.

Values

`tx_layer1` The first layer on which the constraint is applied. The shape on this layer is the enclosing shape.

Type: String (layer and purpose names) or Integer (layer number)

`tx_layer2` The second layer on which the constraint is applied. The shape on this layer is enclosed by the `layer1` shape.

Type: String (layer and purpose names) or Integer (layer number)

`f_ext f_oppExt`

The extension values.

- `f_ext`: The minimum extension on opposite sides in one direction, horizontal or vertical.
- `f_oppExt`: The minimum extension on opposite sides in the perpendicular direction.

`"width" nil nil`

This identifies the index for `table`.

Virtuoso Technology Data Constraints Reference

Extension Constraints

g_table

The *table* row is defined as follows:

```
( f_width ((f_ext1 f_oppExt1) (f_ext2  
f_oppExt2)...)
```

where,

- *f_width* is the width of the *layer1* shape. The constraint applies only if the width of the *layer1* shape is greater than or equal to this value.
- *f_ext1* *f_oppExt1* *f_ext2* *f_oppExt2* are the extension pairs.

Type: A 1-D table specifying floating-point width and extension values

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

`'cutClass { f_width | (f_width f_length) | t_name }`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

`'endSide`

The first extension pair is applied to the short edge of the cut shape and the second extension pair is applied to the long edge of the cut shape (applies only to rectangular cut shapes).

Type: Boolean

`'offCenterline`

The constraint applies only to rectangular cut shapes that do not collide with the centerline of the enclosing shape.

Type: Boolean

`'cutDistance f_cutDistance`

The constraint does not apply if a cut shape has a neighboring cut shape at a distance less than or equal to this value.

`'noSharedEdge`

The constraint does not apply if the two cut shapes that satisfy `'cutDistance` are placed along different edges of the enclosing shape.

In other words, if the two cut shapes are placed along the same edge of the enclosing shape, the extra cut shape along the edge is not counted and the constraint applies.

Type: Boolean

`'hasParaRunLength`

The constraint does not apply to cut shapes that are at a distance less than or equal to *cutDistance* if cut edges opposite all failing cut edges (edges that do not satisfy the minimum extension requirement) share a parallel run length greater than zero with neighboring cut shapes.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'hasExactPrl f_hasExactPrl`

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply to cut shapes that are at a distance less than or equal to `cutDistance` if the parallel run length between the cut shapes is equal to this value.

`'length f_length`

The constraint applies only if the length of the enclosing shape is greater than or equal to this value.

`'extraCut`

The constraint applies only if two or more cut shapes overlap a contiguous same-metal shape.

For via cuts, `'extraCut` can be used without specifying `'cutDistance`. However, for raw cuts, you must specify `'cutDistance`.

Type: Boolean

`'extraOnly`

(Virtuoso Advanced Node for Layout Only) If a `minOppExtension` constraint definition contains this parameter, it is this constraint definition that must be satisfied, and not the other extension rules that do not depend on width. Width-dependent extension rules still apply.

Type: Boolean

`'exactPrl f_exactPrl`

(Virtuoso Advanced Node for Layout Standard) The constraint applies to a via with two or more cuts only if the cuts have parallel run length equal to this value.

`'redundantCutWithin f_distance`

The constraint applies only if the cut shape is at a distance less than or equal to this value from a cut shape that satisfies another `minOppExtension` constraint defined without the `'redundantCutWithin` parameter. Additionally, the two cut shapes must overlap a contiguous same-metal shape on the layer above or below.

The constraint defined with this parameter has smaller enclosure requirements.

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'horizontal | 'vertical`

(Virtuoso Advanced Node for Layout Standard) The first extension pair is measured in the given direction; the second extension pair is measured in the direction perpendicular to the given direction. If direction is not specified, the extension pairs are measured in any direction.

Type: Boolean

`'widthHorizontal | 'widthVertical`

(Virtuoso Layout Suite EXL and higher tiers) Only widths of metal shapes in the specified direction are considered for the corresponding width-based extension lookup.

`'exceptCutMetalEdgeExt | 'useMinWidth`

When the `exceptCutMetalEdgeExt` parameter is used, if an edge of a cut shape or an edge of a metal shape enclosing the cut shape protrudes over a wide wire, it is exempted from the extension requirement, and the constraint value based on the width of the wide wire applies to the other three edges that overlap the wide wire. The cut shape must also meet the extension requirement that applies based on the width of the enclosing metal shape.

Alternatively, use the `useMinwidth` parameter to specify that the minimum width of the wires that overlap a cut should be used to look up the corresponding width-based extension.

Type: Boolean

`'includeAbutted`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the cut shape abuts a wider shape.

Type: Boolean

`'area f_area`

The constraint does not apply if the area of the enclosing shape is greater than this value.

`'minCorner`

Only the minimum extension is required on the corners.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'stepSizePair (f_val1 f_val2)`

(Virtuoso Advanced Node for Layout Standard) The extensions are applied as multiples of these two values. The order of the two `'stepSizePair` values corresponds to the order of the constraint values. If an entry is zero, no stepped extension applies in that direction.

The required extension is calculated as follows:

constraint value + $n \cdot \text{stepSizePair}$, where $n \geq 0$

`'jogLengthOnly f_jogLength`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if a cut shape overlaps with a wrong-way jog—with edge length less than `jogLength` and width equal to `jogWireWidth`—sandwiched by two right-way wires with projected parallel run length less than zero in the wrong-way direction in a 'Z' shape.

`'treatLAsJog`

(Virtuoso Advanced Node for Layout Only) The constraint also applies to an L-shaped jog.

Type: Boolean

`'horizontalJog | 'verticalJog`

(Virtuoso Advanced Node for Layout Only) The direction in which the length of the jog is measured.

Type: Boolean

`'jogWireWidth f_jogWireWidth`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the width of a Z-shaped jog is equal to this value.

`'hollowHorizontal f_horiLength | 'hollowVertical f_vertLength`

(Virtuoso Advanced Node for Layout Only) The portion of the via cut that does not require an extension.

- `'hollowHorizontal`: The center of the via cut extended horizontally by the specified value on both sides does not require an extension.
- `'hollowVertical`: The center of the via cut extended vertically by the specified value on both sides does not require an extension.

Virtuoso Technology Data Constraints Reference

Extension Constraints

'hollowOffset *f_offset*

(Virtuoso Layout Suite EXL and higher tiers) The exact offset for the hollow portion to be misaligned from the center of the via. The default is 0.

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
```

(Virtuoso Advanced Node for Layout Standard) Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- 'outsideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Standard) The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Standard) The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'distanceWithin f_distance 'nearbyLayer tx_layer
```

The constraint applies only if the distance between the *layer1* shape (metal shape) and the *layer* shape, in Euclidean measurement, is less than or equal to *distance*.

Type: String (layer name) or Integer (layer number)

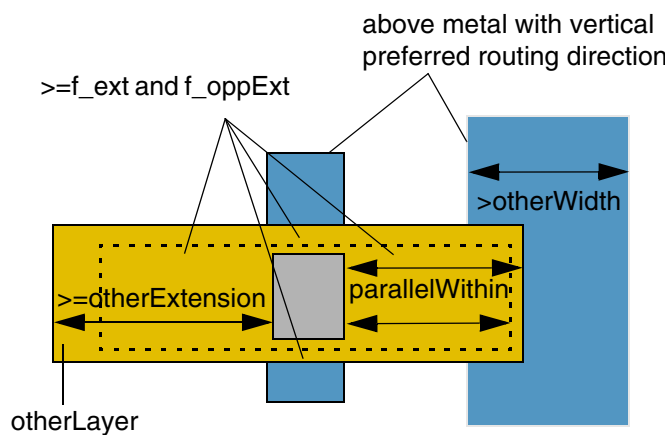
Virtuoso Technology Data Constraints Reference

Extension Constraints

```
'otherLayer tx_otherLayer 'otherWidth f_otherWidth  
{'withinHorizontal | 'withinVertical} 'parallelWithin  
f_parallelWithin ['otherExtension f_otherExtension]
```

(Virtuoso Advanced Node for Layout Only) The constraint applies only if there is a shape on layer *tx_otherLayer* with width greater than *f_otherWidth* within *f_parallelWithin* measured in direction *withinHorizontal* or *withinVertical* of this cut shape but on a different net.

(Virtuoso Layout Suite EXL and higher tiers) The *f_otherExtension* value specifies the enclosure on the opposite side of a cut edge that faces the wire fulfilling the *f_otherWidth* condition.



f_default

The extension value to be used when no table entry applies.

'coincidentAllowed

The edges of a cut shape can coincide with the edges of the enclosing shape. Otherwise, the cut shape must meet the extension requirement.

Type: Boolean

```
'alignedLayer g_layer 'alignedHorizontal | 'alignedVertical
```

(Virtuoso Advanced Node for Layout Standard) The layer for which the alignment of the via with the shape is to be checked. Alignment is checked in the direction specified.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Examples

- [Example 1: minOppExtension with endSide](#)
- [Example 2: minOppExtension with vertical](#)
- [Example 3: minOppExtension with offCenterline](#)
- [Example 4: minOppExtension with includeAbutted](#)
- [Example 5: minOppExtension with cutDistance](#)
- [Example 6: minOppExtension with cutDistance and noSharedEdge](#)
- [Example 7: minOppExtension with cutDistance and hasParaRunLength](#)
- [Example 8: minOppExtension with extraCut](#)
- [Example 9: minOppExtension with redundantCutWithin](#)
- [Example 10: minOppExtension with exceptCutMetalEdgeExt](#)
- [Example 11: minOppExtension with length](#)
- [Example 12: minOppExtension with jogLengthOnly, verticalJog, and jogWireWidth](#)
- [Example 13: minOppExtension with hollowHorizontal and hollowOffset](#)
- [Example 14: minOppExtension with distanceWithin and nearbyLayer](#)
- [Example 15: minOppExtension with otherExtension](#)
- [Example 16: minOppExtension with widthHorizontal and widthVertical](#)
- [Example 17: minOppExtension with alignedLayer](#)

Virtuoso Technology Data Constraints Reference

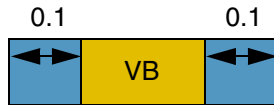
Extension Constraints

Example 1: *minOppExtension with endSide*

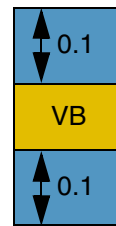
A Via1 via cut of type VB (0.4x0.6) enclosed by a Metal1 wire must have 0.1 extensions on the short edges and 0.0 extensions on the long edges of the enclosing wire.

```
orderedSpacings (  
  ( minOppExtension "Metal1" "Via1"  
    'cutClass "VB"  
    'endSide  
    (0.1 0.0)  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1



a) PASS. The extensions are 0.1 on the short edges and zero on the long edges.



b) FAIL. The extension must be 0.1 on the short edges and zero on the long edges.

Virtuoso Technology Data Constraints Reference

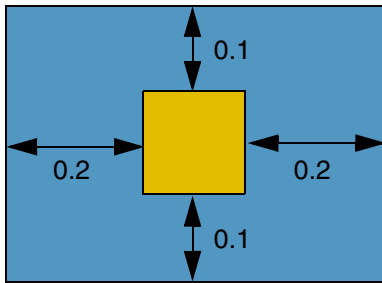
Extension Constraints

Example 2: minOppExtension with vertical

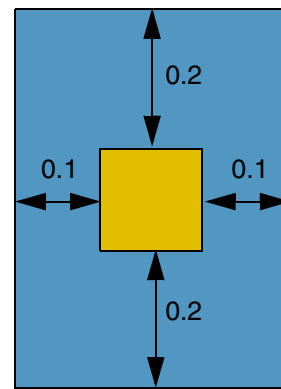
A Via1 via cut enclosed by a Metal1 wire must have 0.1 extensions in the vertical direction and 0.2 extensions in the horizontal direction.

```
orderedSpacings(  
  ( minOppExtension "Metal1" "Via1"  
    'vertical  
    (0.1 0.2)  
  )  
) ;orderedSpacings
```

■ Via1
■ Metal1



a) PASS. The extensions are 0.1 in the vertical direction and 0.2 in the horizontal direction.



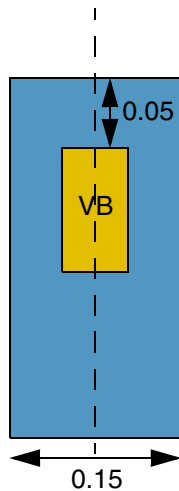
b) FAIL. The extensions in the vertical direction must be 0.1. The constraint would pass if the direction was not specified.

Example 3: minOppExtension with offCenterLine

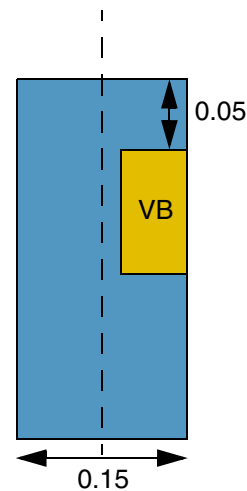
A Via1 via cut of type VB (0.4x0.6) enclosed by a Metal1 wire must have 0.05 extension in one direction and 0.03 extension in the perpendicular direction if the width of the Metal1 wire is greater than or equal to 0.15, provided the via cut does not collide with the centerline of the wire.

```
spacingTables (
  ( minOppExtension "Metal1" "Via1"
    (( "width" nil nil )
      'cutClass "VB"
      'offCenterLine
    )
    ( 0.15 ((0.05 0.03)) )
  )
) ;spacingTables
```

■ Via1
■ Metal1



a) The constraint does not apply because the via cut collides with the centerline of the wire that is 0.15 wide.



b) FAIL. The via cut does not collide with the centerline of the 0.15 wide wire. Therefore, the constraint applies. However, the right edge of the via cut does not have a 0.03 extension.

Virtuoso Technology Data Constraints Reference

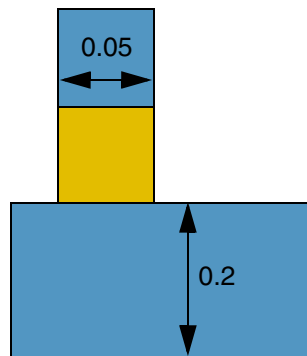
Extension Constraints

Example 4: minOppExtension with includeAbutted

A Via1 via cut enclosed by a Metal1 wire with width greater than or equal to 0.2 must have 0.05 extensions in both directions if it abuts a wider wire.

```
spacingTables(  
  ( minOppExtension "Metal1" "Via1"  
    (( "width" nil nil )  
      'includeAbutted  
    )  
    ( 0.2 ((0.05 0.05)) )  
  )  
) ;spacingTables
```

■ Via1
■ Metal1



FAIL. The via cut is abutted to a wide wire, but does not have the required 0.05 extensions on all sides. The constraint would not apply if 'includeAbutted was not specified.

Virtuoso Technology Data Constraints Reference

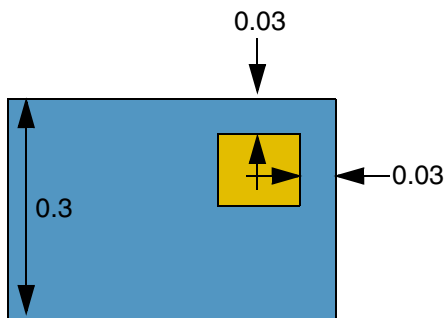
Extension Constraints

Example 5: *minOppExtension* with *cutDistance*

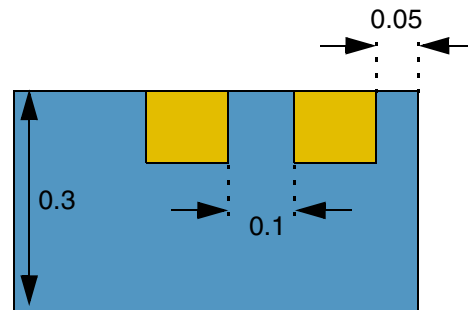
A Via1 via cut enclosed by a Metal1 wire with width greater than or equal to 0.3 must have 0.03 extensions on all sides. If two via cuts are less than 0.2 apart, the constraint does not apply.

```
spacingTables (  
  ( minOppExtension "Metal1" "Via1"  
    (( "width" nil nil )  
      'cutDistance 0.2  
    )  
    ( 0.3 ((0.03 0.03)) )  
  )  
);spacingTables
```

■ Via1
■ Metal1



a) PASS. The via cut has 0.03 extensions on all sides.



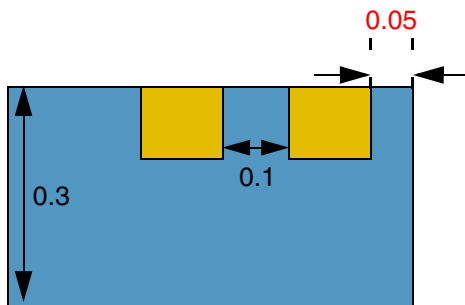
b) The constraint does not apply because the distance between the via cuts is 0.01 (<0.02).

Example 6: *minOppExtension* with *cutDistance* and *noSharedEdge*

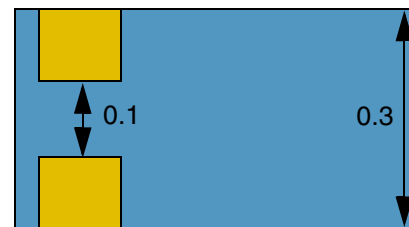
A Via1 via cut enclosed by a Metal1 wire with width greater than or equal to 0.3 must have 0.03 extensions on all sides. If two via cuts are less than 0.2 apart and are placed along different edges of the wire, the constraint does not apply.

```
spacingTables (
  ( minOppExtension "Metal1" "Via1"
    (( "width" nil nil )
      'cutDistance 0.2 'noSharedEdge
    )
    ( 0.3 ((0.03 0.03)) )
  )
) ;spacingTables
```

■ Via1
■ Metal1



a) FAIL. The distance between the via cuts is 0.01 (<0.02) and they are both along the same edge of the wire. Because 'noSharedEdge' is specified, the extra cut is ignored and the extension requirement applies, but is not met.



b) The constraint does not apply because the via cuts are placed along different edges of the wire.

Example 7: *minOppExtension* with *cutDistance* and *hasParaRunLength*

If the width of a Metal1 shape is greater than or equal to 0.3, Via1 via cuts must have 0.03 extensions on all sides. This width-based constraint definition does not apply if:

- The distance between two via cuts is less than 0.2.
- The cut edges that do not meet the extension requirement have parallel run length greater than zero with neighboring via cuts on the opposite side.

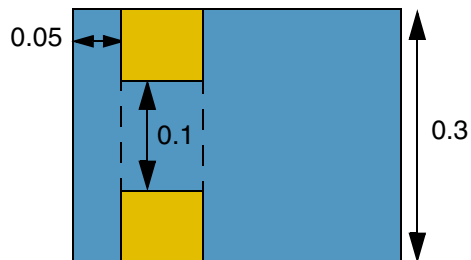
Virtuoso Technology Data Constraints Reference

Extension Constraints

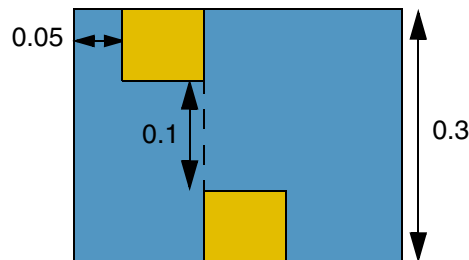
When the width-based constraint does not apply, an extension requirement of (0.0, 0.05) or (0.2, 0.2) must be met.

```
orderedSpacings(  
  ( minOppExtension "Metal1" "Via1"  
    (0.0 0.5)  
  )  
  ( minOppExtension "Metal1" "Via1"  
    (0.2 0.2)  
  )  
) ;orderedSpacings  
spacingTables(  
  ( minOppExtension "Metal1" "Via1"  
    (( "width" nil nil )  
      'cutDistance 0.2 'hasParaRunLength  
    )  
    ( 0.30 ((0.03 0.03)) )  
  )  
) ;spacingTables
```

■ Via1
■ Metal1



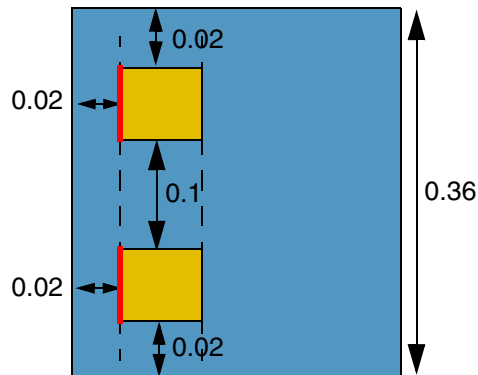
a) PASS. The width of the Metal1 shape is 0.3; therefore, extension requirement on all sides is 0.03. This extension requirement is not met on the topmost and bottommost cut edges, but these cut edges are exempt because the distance between the two via cuts is 0.1 (< 0.2) and the parallel run length between the via cuts is greater than zero (that is, the width-based constraint definition does not apply). Both via cuts meet the (0.0, 0.05) extension requirement.



b) FAIL. The width of the Metal1 shape is 0.3; the distance between the two via cuts is 0.1 (< 0.2), but the parallel run length between them is zero. Therefore, the width-based constraint applies, but extension requirement of 0.03 is not met on all sides.

Virtuoso Technology Data Constraints Reference

Extension Constraints



c) FAIL. The width of the Metal1 shape is 0.36; therefore, extension requirement on all sides is 0.03. This extension requirement is not met on the topmost, bottommost, and two left cut edges. The topmost and bottommost cut edges are exempt because the distance between the two via cuts is 0.1 (< 0.2) and the parallel run length between the via cuts is greater than zero. However, there do not exist neighboring via cuts that have parallel run length greater than zero with the edges that are opposite the left failing edges (in red).

Example 8: minOppExtension with extraCut

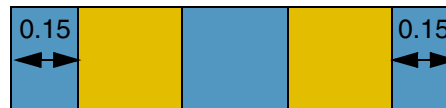
If two or more Via1 via cuts overlap a contiguous Metal1 shape, the via cuts must have 0.0 extension in one direction and 0.15 extension in the perpendicular direction.

```
orderedSpacings (
  ( minOppExtension "Metal1" "Via1"
    'extraCut
    (0.00 0.15)
  )
) ;orderedSpacings
```

■ Via1
■ Metal1



a) The constraint does not apply because there is only one via cut.



b) PASS. The number of via cuts overlapped by a single metal shape is 2, and these via cuts have 0.15 extensions on opposite sides.

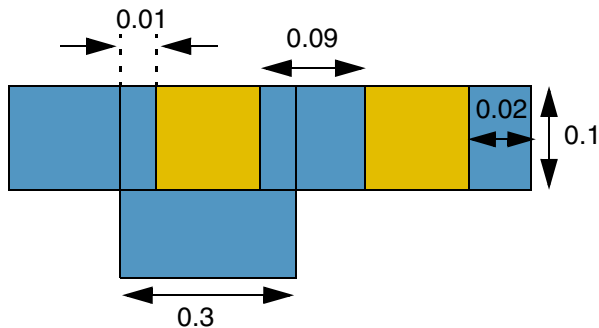
Example 9: minOppExtension with redundantCutWithin

A Via1 via cut enclosed by a Metal1 shape with width greater than or equal to 0.3 must have 0.03 extensions on all sides. If two or more via cuts overlap a contiguous Metal1 shape, the via cuts must have 0.0 extension in one direction and 0.02 extension in the perpendicular direction. Additionally, a via cut placed less than 0.1 apart from another via cut must have 0.0 extension in one direction and 0.01 extension in the perpendicular direction.

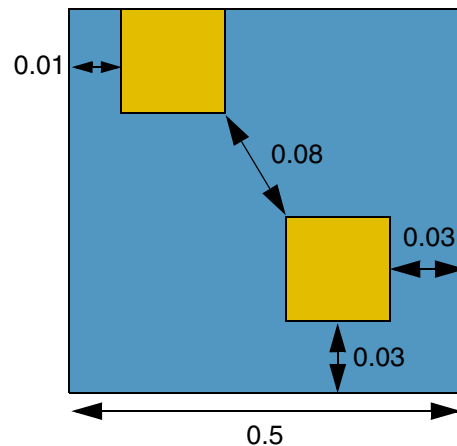
```
orderedSpacings (
  ( minOppExtension "Metal1" "Via1"
    'extraCut
    (0.00 0.02)
  )
  ( minOppExtension "Metal1" "Via1"
    'redundantCutWithin 0.1
    (0.00 0.01)
  )
) ;orderedSpacings

spacingTables (
  ( minOppExtension "Metal1" "Via1"
    (( "width" nil nil ))
    ( 0.3 ((0.03 0.03)) )
  )
) ;spacingTables
```

■ Via1
■ Metal1



a) PASS. The two via cuts are overlapped by a single metal shape, and the right via cut has a 0.02 extension. Therefore, 'extraCut is satisfied. The left via cut is 0.09 (<0.1) away from the right via cut and has 0.01 horizontal extension and zero vertical extension. Therefore, 'redundantCutWithin is also satisfied.



b) PASS. The bottom via cut enclosed by a 0.5 (>0.3) wide wire has vertical and horizontal extensions of 0.03. The top via cut is 0.08 (<0.1) away from the bottom via cut and has 0.01 horizontal extension and zero vertical extension.

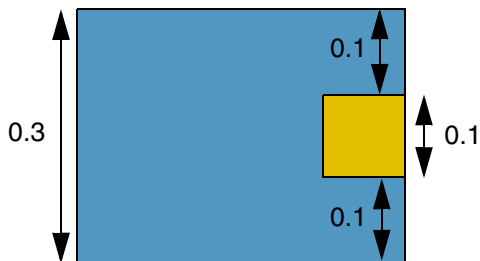
Example 10: minOppExtension with exceptCutMetalEdgeExt

If an edge of a Via1 via cut or an edge of a Metal1 shape enclosing the via cut protrudes over a wide wire, it is exempted from the extension requirement. The following are the extension requirements based on width:

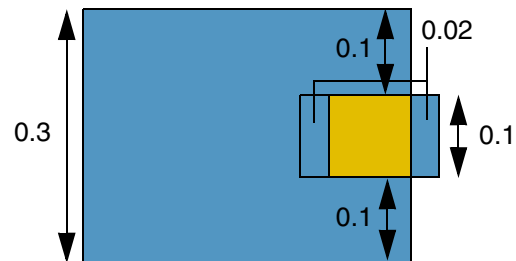
- A via cut enclosed by a Metal1 shape with width greater than or equal to 0.3 must have 0.03 extensions on all sides.
- A via cut enclosed by a Metal1 shape with width greater than or equal to 0.15 must have 0.02 extensions on all sides.

```
orderedSpacings (
  ( minOppExtension "Metal1" "Via1"
    (0.00 0.02)
  )
) ;orderedSpacings
spacingTables (
  ( minOppExtension "Metal1" "Via1"
    (( "width" nil nil )
      'exceptCutMetalEdgeExt
    )
    (
      0.15 ((0.02 0.02))
      0.30 ((0.03 0.03))
    )
  )
) ;spacingTables
```

■ Via1
■ Metal1



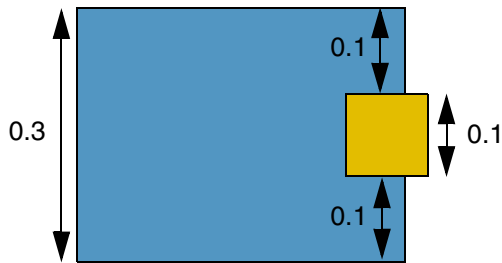
a) FAIL. The width of the Metal1 shape is 0.3, but the right edge of the via cut does not have a 0.03 extension.



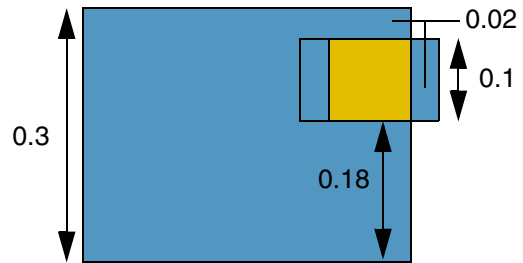
b) PASS. The protrusion on the right side exempts that edge of the via cut, and the other three edges have 0.03 extensions. Basic extension requirement of (0.0, 0.02) is also met.

Virtuoso Technology Data Constraints Reference

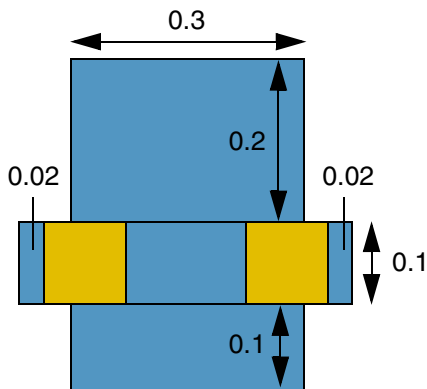
Extension Constraints



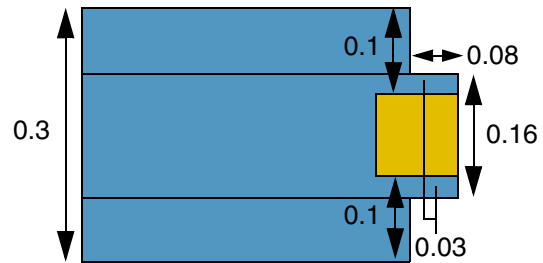
c) FAIL. The protrusion on the right side exempts that edge of the cut shape, and the other three edges have 0.03 extensions. However, the basic extension requirement of (0.0, 0.02) is not met.



d) FAIL. The protrusion on the right side exempts that via cut edge. However, the top edge has an extension of only 0.02 (< 0.03). Basic extension requirement of (0.0, 0.02) is met.



e) PASS. The protrusions on the left and right sides exempt those via cut edges, and the other three via cut edges have 0.03 extensions. Basic extension requirement of (0.0, 0.02) is also met.



f) FAIL. The width of the Metal1 shape that encloses the via cut is 0.16. This requires extensions of 0.02 on all sides, a condition that is not satisfied for the right edge of the via cut.

Virtuoso Technology Data Constraints Reference

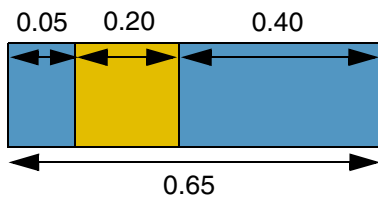
Extension Constraints

Example 11: minOppExtension with length

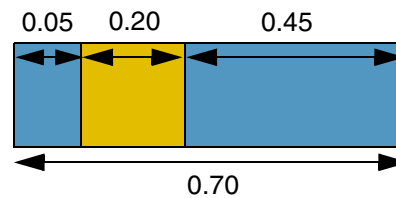
A Via1 via cut enclosed by a Metal1 wire with length greater than or equal to 0.70 must have 0.05 extension in one direction and 0.00 extension in the perpendicular direction.

```
orderedSpacings(
  ( minOppExtension "Metal1" "Via1"
    'length 0.70
    (0.05 0.00)
  )
);orderedSpacings
```

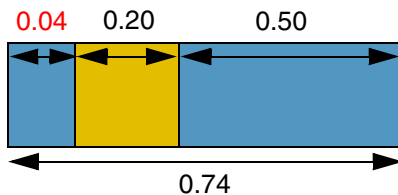
■ Via1
■ Metal1



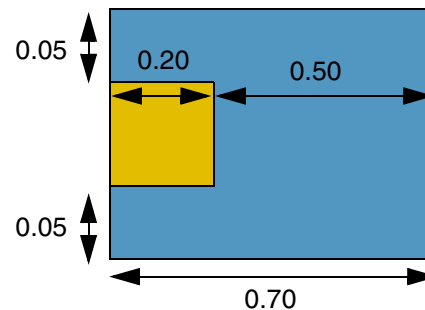
a) The constraint does not apply because the length of the wire is 0.65 (<0.70).



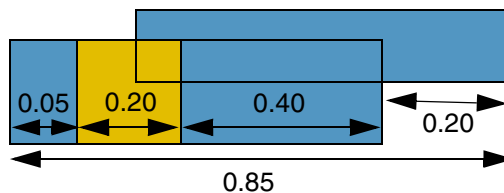
b) PASS. The length of the wire is 0.70, and the horizontal extension is 0.05 and the vertical extension is 0.



c) FAIL. The length of the wire is 0.74 (>0.70), but the horizontal extension is 0.04 (<0.05).



d) PASS. The length of the wire is 0.70, and the vertical extensions are 0.05 and the horizontal extension is 0.



e) PASS. The length of the wire is 0.85 (>0.70), and the horizontal extension is 0.05 and the vertical extension is 0.

Virtuoso Technology Data Constraints Reference

Extension Constraints

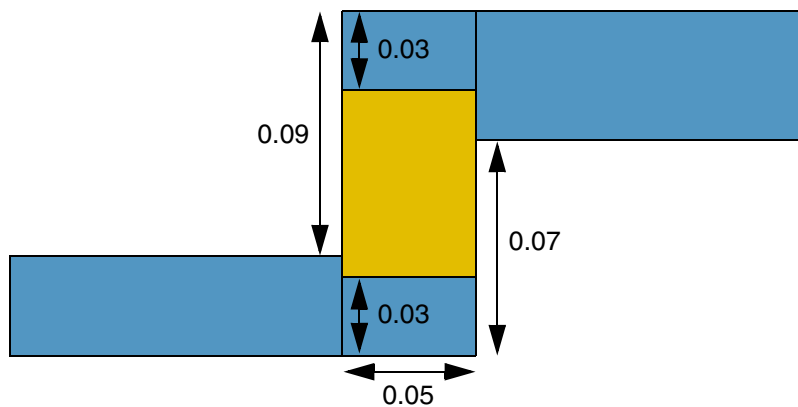
Example 12: minOppExtension with jogLengthOnly, verticalJog, and jogWireWidth

A Via1 via cut enclosed by a Metal1 wrong-way Z-shaped jog with edges less than 0.1 long and width equal to 0.05 must have 0.03 extension in one direction and 0.00 extension in the perpendicular direction.

```
spacingTables (
  ( minOppExtension "Metal1" "Via1"
    (( "width" nil nil )
      'jogLengthOnly 0.1
      'verticalJog
      'jogWireWidth 0.05
    )
    ( 0.00 ((0.00 0.03)) )
  )
) ;spacingTables
```

■ Via1
■ Metal1

Preferred routing direction: Horizontal



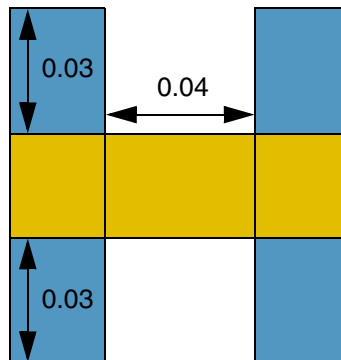
PASS. The via cut is overlapped by a wrong-way Z-shaped jog. Both edges of the jog are less than 0.1 long and the width of the jog is equal to 0.05. The horizontal extension is 0 and the vertical extension is 0.03.

Example 13: minOppExtension with hollowHorizontal and hollowOffset

A Via1 via cut enclosed by a Metal1 shape must have 0.03 extension in one direction and 0.00 extension in the perpendicular direction. The center of the via cut extended horizontally by 0.02 on both sides, with an offset of 0.005, does not require an extension.

```
spacingTables (
  ( minOppExtension "Metal1" "Via1"
    (( "width" nil nil )
      'hollowHorizontal 0.02 'hollowOffset 0.005
    )
    ( 0.00 ((0.00 0.03)) )
  )
) ;spacingTables
```

■ Via1
■ Metal1



PASS. The middle portion of the via cut with width 0.04 (the center of the via cut extended horizontally by 0.02 on both sides) does not require an extension.

Example 14: minOppExtension with distanceWithin and nearbyLayer

A V1 via cut enclosed by an M1 shape must have 0.1 extension in both directions if the distance between the M1 shape and a nearby MIMCAP shape is less than or equal to 15.0. Otherwise, the extension in both directions must be 0.05.

```
orderedSpacings (
  ( minOppExtension "M1" "V1"
    (0.05 0.5)
  )
  ( minOppExtension "M1" "V1"
    'distanceWithin 15.0 'nearbyLayer "MIMCAP"
    (0.1 0.1)
  )
) ;orderedSpacings
```

Example 15: minOppExtension with otherExtension

The constraint applies to a Via1 cut shape enclosed by Metal1 when a vertical Metal2 shape of width 0.015 or greater is within 0.025 of the cut shape. The extension must be at least 0 in one direction and 0.02 in the other direction. In addition, the extension in the direction away from the Metal2 wire must be 0.023.

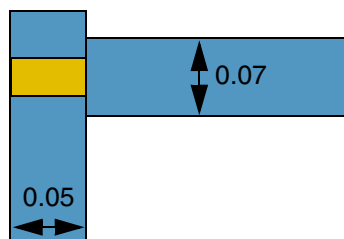
```
orderedSpacings (
  ( minOppExtension "M1" "V1"
    'otherLayer "Metal2"
    'otherWidth 0.015
    'withinHorizontal
    'parallelWithin 0.025
    'otherExtension 0.023
    (0.0 0.02)
  )
) ;orderedSpacings
```

Example 16: minOppExtension with widthHorizontal and widthVertical

Consider the following constraint when width directions are specified or not specified.

```
orderedSpacings (
  ( minOppExtension "Metal1" "Via1"
    (("width" nil nil ) 'endSide)
    (
      .05 ((0 0.03))
      .07 ((0.04 0.04))
      .09 ((0.06 0.06))
    )
  )
) ;orderedSpacings
```

Via1
 Metal1



FAIL, if width direction is not specified.
 PASS, if widthHorizontal is specified.
 FAIL, if widthVertical is specified.

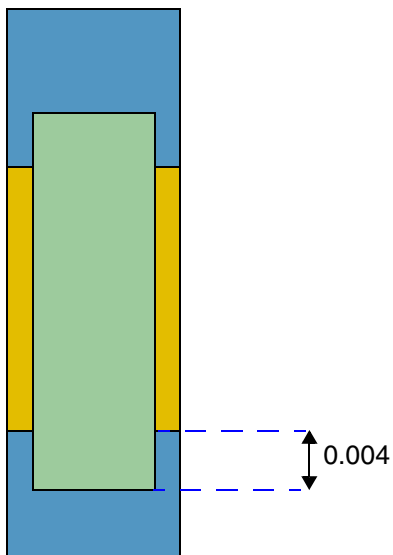
Example 17: *minOppExtension* with *alignedLayer*

Consider the following constraints specified in an OR group, where only the first constraint specifies *alignedLayer*.

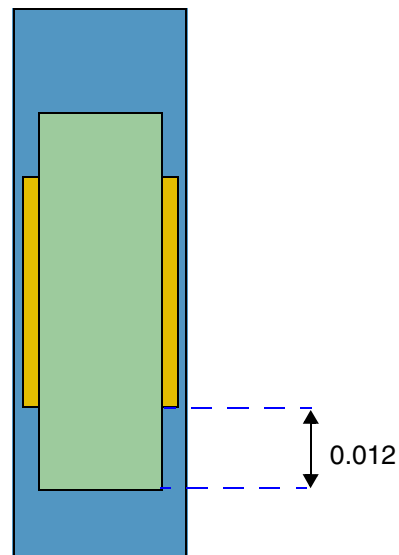
```
spacingTables(
  ( minOppExtension "Metal2" "Via1"
    (( "width" nil nil )
      'cutClass (0.016 0.032)
      'endSide
      'alignedLayer "Metal1" 'alignedVertical)
    ( 0.015 ((0.004 -0.002)) )
  )
);spacingTables

spacingTables(
  ( minOppExtension "Metal2" "Via1"
    (( "width" nil nil )
      'cutClass (0.016 0.032)
      'endSide
      ( 0.015 ((0.012 -0.002)) )
    )
  )
);spacingTables
```

Via1
 Metal1
 Metal2



The first constraint applies because the via is self-aligned.

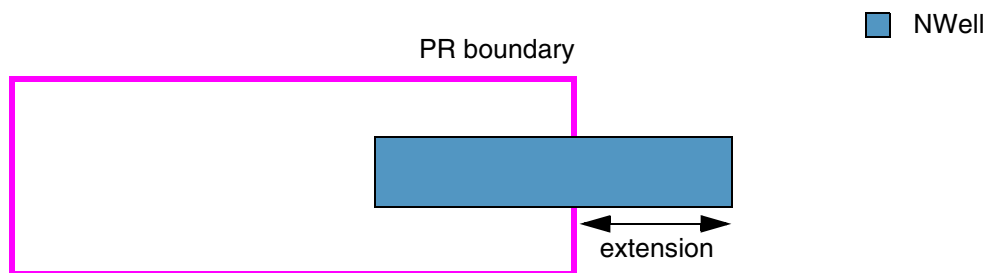


The second constraint applies because the via is not self-aligned.

minPRBoundaryExtension

```
spacings (
    ( minPRBoundaryExtension tx_layer
      f_extension ['coincidentAllowed']
    )
) ;spacings
```

Specifies the minimum distance a shape must extend past the PR boundary on a layer. This constraint is commonly used to describe the necessary extension of a well past a boundary.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension of the shape past the PR boundary must be greater than or equal to this value.

Parameters

'coincidentAllowed'	The shape can coincide with the PR boundary. Otherwise, the spacing requirement must be met. Type: Boolean

Virtuoso Technology Data Constraints Reference

Extension Constraints

Example

The extension of a NWell shape past the PR boundary must be at least 1.5.

```
spacings (
  ( minPRBoundaryExtension "NWell"
    1.5
  )
) ;spacings
```

minProtrusionWidthWithVia

```
orderedSpacings (
  ( minProtrusionWidthWithVia tx_metalLayer tx_cutLayer
    'widthLengthTable ((f_wideWidth f_length) ...)
    'cutClass {f_width | (f_width f_length) | t_name}
    f_width
  )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum width and length for a protrusion on a wide wire. The length of the protrusion varies based on the width of the wide wire. The constraint applies only if the protrusion contains a single via cut of the specified cut class.

Values

<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_width</i>	The width of the protrusion must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

'widthLengthTable (*f_wideWidth* *f_length*)

If the width of the wide wire on which the protrusion exists is greater than or equal to *wideWidth*, the length of the protrusion must be greater than or equal to *length*.

'cutClass {*f_width* | (*f_width* *f_length*) | *t_name*}

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

The constraint applies only if the protrusion contains a single via cut of this cut class.

Example

If the protrusion on a wide Metal1 wire contains a single via cut of type VA on Via1, the following conditions must be met:

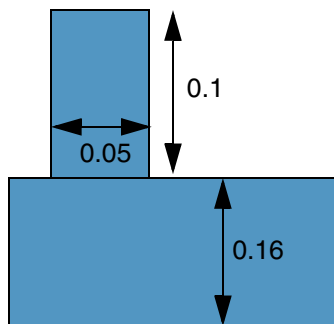
- The width of the protrusion must be at least 0.05.
- If the width of the wide wire is greater than or equal to 0.11, the length of the protrusion must be at least 0.12.

Virtuoso Technology Data Constraints Reference

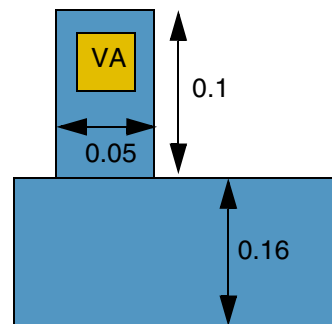
Extension Constraints

```
orderedSpacings (
  ( minProtrusionWidthWithVia "Metal1" "Via1"
    'widthLengthTable ((0.11 0.12))
    'cutClass "VA"
    0.05
  )
) ;orderedSpacings
```

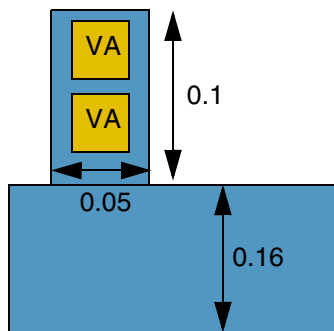
Via1
 Metal1



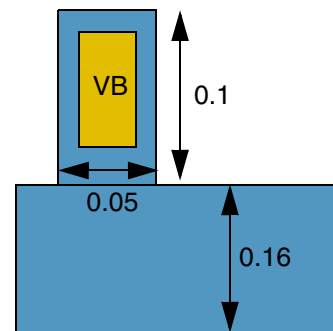
a) The constraint does not apply because the protrusion does not enclose a via cut.



b) FAIL. The protrusion contains a single VA via cut and is 0.05 wide, but the length of the protrusion is only 0.1 (< 0.12).



c) The constraint does not apply because the protrusion contains more than one via cut.



d) The constraint does not apply because the protrusion encloses a via cut of type VB (and not VA).

minQuadrupleExtension

```
spacingTables(
  ( minQuadrupleExtension tx_layer1 tx_layer2
    (( "width" nil nil )
      ['cutClass {f_width | (f_width f_length) | t_name}]
      ['allSides
        ['useMaxWidth | 'useMinWidth | 'withinFirstWidth]
      ]
      ['horizontal | 'vertical]
      ['mask1 | 'mask2 | 'mask3]
      ['directionalExtension]
      ['otherLayer tx_otherLayer
        ['parallelWithinTable g_paraWithinTable]
        ['otherMask1 | 'otherMask2 | 'otherMask3]
        | 'trimLayer tx_trimLayer ['trimLengthTable g_trimLengthTable]
        | 'exactZero g_exactZeroTable
          ['includeAbuttedTable g_includeAbuttedTable]
      ]
      ['otherWithinWidthTable g_otherWithinWidthTable
        {'otherWithinWidthHorizontal | 'otherWithinWidthVertical}
        ['otherSideTable g_otherSideTable]
        ['exceptMetalOverlap]
      ]
      ['lineEndGapOnlyTable g_lineEndGapOnlyTable]
      ['mustUseTable g_mustUseTable]
      ['lengthTable g_lengthTable]
      ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum extension of a shape with the specified minimum width on *layer1* past a shape on *layer2*. The extension of a shape on *layer1* can have up to four different values, applicable to the four edges of the *layer2* shape, which is typically a square or a 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 other pair of opposite edges.

When *layer1* is a non-via layer and *layer2* a via layer, an enclosure check is done; *layer2* when expanded by constraint values should be enclosed by *layer1* in this case.

Virtuoso Technology Data Constraints Reference

Extension Constraints

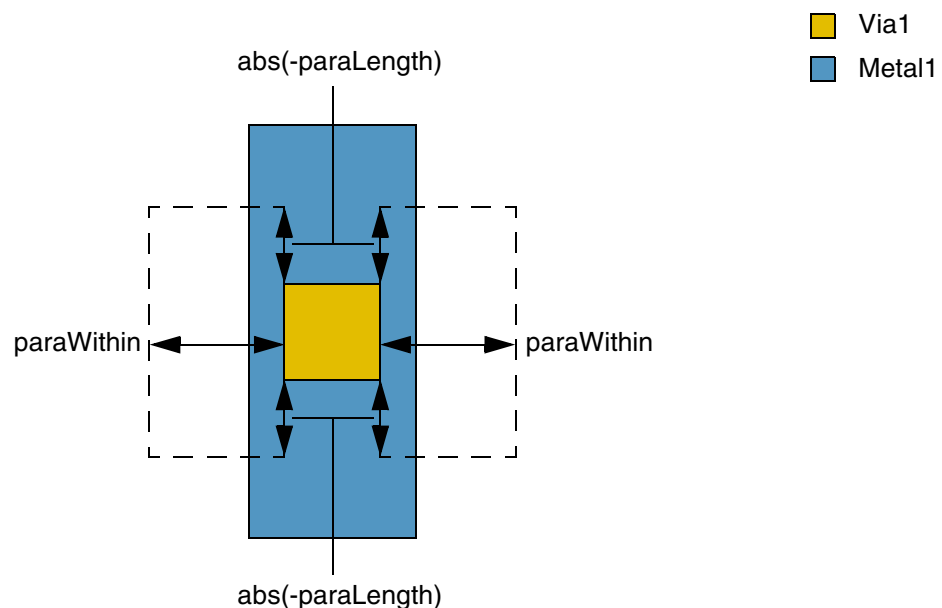
When both layers are non-via layers, an extension check is done from the inner edge of *layer1* to the outer edge of *layer2*.

Optionally, if either of the two extension sets is asymmetric, that is, *extension1* is not equal to *extension2*, or *extension3* is not equal to *extension4*, the smallest of the four extension values must be met on all four sides of the *layer2* shape, including the corners. Additionally, in this case, the extensions apply to the minimum width *layer1* shape that the *layer2* shape touches or overlaps. If both extension sets are symmetric, that is, *extension1* equals *extension2* and *extension3* equals *extension4*, the specified extensions apply to the maximum width *layer1* shape that the *layer2* shape overlaps or touches.

The `minQuadrupleExtension` constraint can be set to apply only in a certain direction. In this case, the extensions in the specified direction must meet the first constraint-value-pair, and the extensions in the perpendicular direction must meet the second constraint-value-pair.

However, if `'directionalExtension` is specified, the first two extension values are applied on the short edges of a rectangular via cut, while the last two extension values are applied on the long edges of the rectangular via cut.

If `'parallelWithinTable` is specified, the constraint applies only if there exists a *layer1* (metal) neighbor on at least one side of the via cut at a distance less than *paraWithin* from it and has common parallel run length greater than or equal to *paraLength* with it (that is, overlaps or is inside a dotted search window shown in the figure below).



Virtuoso Technology Data Constraints Reference

Extension Constraints

An OR constraint group should contain settings for only the same constraint, with different parameters. If any of the settings is satisfied, the constraint requirement is met. An exception is made for `minQuadrupleExtension`, `minOppExtension`, and `minExtensionDistance` constraints, which can be bound together in an OR group, and if any of these constraints is satisfied, the extension requirement is met.

Values

<code>tx_layer1</code>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>tx_layer2</code>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>"width" nil nil</code>	This identifies the index for <i>table</i> .
<code>f_default</code>	The extension value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

Extension Constraints

g_table

The *table* row is defined as follows:

```
( f_width ((f_ext1 f_ext2 f_ext3 f_ext4  
['minSum'] ['maxLength g_maxLength])) ... )
```

where,

- *f_width* is the width of the *layer1* shape. The constraint applies only if the width of the *layer1* shape is greater than or equal to this value.
- *f_ext1* *f_ext2* *f_ext3* *f_ext4* are extension pairs. The first pair specifies *layer1* extensions on one set of opposite sides/edges of the *layer2* shape, and the second pair specifies extensions on the other set of opposite sides/edges of the *layer2* shape.

Type: A 1-D table specifying floating-point width and extension values

'minSum: Two sides may 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 of the two specified extensions.

Type: Boolean

'maxLength *f_maxLength*: (Virtuoso Advanced Node for Layout Only) The constraint applies only if the length of the *layer1* shape is less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'allSides`

If either of the extension sets is asymmetric, the smallest of the four extension values must be met on all sides of the cut shape, including the corners.

Type: Boolean

`'useMaxWidth`

(Virtuoso Advanced Node for Layout Only) The width of the widest metal shape that the cut shape overlaps is used to look up the extension requirement.

Type: Boolean

`'useMinWidth`

(Virtuoso Advanced Node for Layout Only) The width of the narrowest metal shape that the cut shape overlaps is used to look up the extension requirement.

Type: Boolean

`'withinFirstWidth`

(Virtuoso Advanced Node for Layout Only) The enclosure required by the first width value specified in the table applies only if the via cut is completely within a wire that satisfies this width. Otherwise, the width of the wire that the via cut touches or overlaps is used to look up the enclosure value.

Type: Boolean

`'horizontal | 'vertical`

The first extension pair is measured in the given direction; the second extension pair is measured in the perpendicular direction. If direction is not specified, the extension pairs are measured in any direction.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Extension Constraints

'mask1 | 'mask2 | 'mask3

(Virtuoso Advanced Node for Layout Only) The constraint applies only to the shapes on the specified mask.

Type: Boolean

'directionalExtension

(Virtuoso Advanced Node for Layout Only) The first two extension values are applied on the short edges of a rectangular via cut, while the last two extension values are applied on the long edges of a rectangular via cut.

Type: Boolean

'otherLayer *tx_otherlayer*

(Virtuoso Advanced Node for Layout Only) The layer against which the width must be checked. However, extension values still apply on the metal to which the constraint applies.

Type: String (layer name) or Integer (layer number)

'parallelWithinTable (*g_paraWithinTable*)

(Virtuoso Advanced Node for Layout Only) The constraint applies only if there exists a *layer1* (metal) neighbor on at least one side of the via cut at a distance less than *f_paraWithin* from it and has common parallel run length greater than or equal to *f_paraLength* with it.

The format of a row in *paraWithinTable* is as follows:

(*f_width* ((*f_paraLength* *f_paraWithin*)))

where, *f_width* is the width of the enclosing metal shape and *f_paraLength* and *f_paraWithin* are the values that apply when width is greater than or equal to *f_width*.

Type: A 1-D table indexed on width, specifying corresponding floating-point parallel run length and distance values

'otherMask1 | 'otherMask2 | 'otherMask3

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies to this mask on *otherLayer*.

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'exactZero g_exactZeroTable`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that if any of the given extension values is equal to zero, the enclosures must be exactly zero on the given sides and more than or equal to the non-zero extension values on the other sides.

`'includeAbuttedTable g_includeAbuttedTable`

(Virtuoso Advanced Node for Layout Only) The table format specified with this parameter must exactly match the table format of the constraint value. In a set of extension values, the first number can be 1 or 0. The value 1 specifies that a cut abutted to a wide wire needs to meet the wide wire extension. The value 0 indicates that the parameter does not apply to this extension.

`'otherWithinWidthTable g_otherWithinWidthTable`

`{'otherWithinWidthHorizontal | 'otherWithinWidthVertical}`

(Virtuoso Layout Suite EXL and higher tiers) The corresponding extension applies only if there is a neighbor wire with width greater than the first value and within a distance given by the second value. The distance is measured in the specified direction (`otherWithinWidthHorizontal` or `otherWithinWidthVertical`).

The table format specified must match the table format of the constraint value. In each set of four numbers corresponding to a set of extensions, the first value is the width of the other wire and the second value is the within search distance of the other wire.

`'otherSideTable g_otherSideTable`

(Virtuoso Layout Suite EXL and higher tiers) The table format specified with this parameter must exactly match the format of the constraint value. For a given extension tuple, if the `otherWithinWidthTable` condition is met, then the extension opposite of the neighbor wire needs to be at least as much as specified in the first number.

`'exceptMetalOverlap`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that the extension requirements specified in the `otherWithinWidthTable` parameter cannot be applied if the cut overlaps with the metal.

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'trimLayer tx_trimLayer`

(Virtuoso Advanced Node for Layout Only) The extension value must be satisfied at the line-end that a trim shape on this layer overlaps.

Type: String (layer name) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'trimLengthTable g_trimLengthTable`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the length of the trim shape is greater than or equal to the first given value in the *trimLengthTable* table, which has the following format:

```
( f_width ((f_trimLength f_trimLength)
(f_trimLength f_trimLength) ...) )
```

where,

- *f_width* is the width of the *layer1* shape. The constraint applies only if the width of the *layer1* shape is greater than or equal to this value.
- *f_trimLength* is the length of the trim shape that touches or overlaps the line-ends of the *layer1* shapes. The constraint applies only if the length of the trim shape is greater than or equal to *f_trimLength*. Additionally, the constraint applies only to the outermost *layer1* shapes that the trim shape touches or overlaps.

Note: For each constraint value in *table*, which comprises a set of four extension values, specify two lists with two *f_trimLength* values each, of which only the first *f_trimLength* value applies and the rest are ignored. As a result, all four *f_trimLength* values in the set can be the same. For a detailed representation, see [Example 7: minQuadrupleExtension with trimLayer and trimLengthTable](#).

Type: A 1-D table indexed on width, specifying corresponding floating-point length value for the trim shape.

`'lineEndGapOnlyTable g_lineEndGapOnlyTable`

(Virtuoso Layout Suite EXL and higher tiers) The constraint value corresponding to this table entry applies only if the [lineEndGap](#) constraint condition on this layer is met. The table format specified must match the table format of the constraint value.

Each constraint value corresponds to two pairs of numbers in this table. The default is 0 for all numbers. The [lineEndGap](#) constraint is enabled by setting the first of the 4 numbers to 1.

`'mustUseTable g_mustUseTable`

Virtuoso Technology Data Constraints Reference

Extension Constraints

(Virtuoso Layout Suite EXL and higher tiers) Specifies if the first value is 1, the corresponding extension set needs to be met and all other extension sets for the same width are ignored.

`'lengthTable g_lengthTable`

(Virtuoso Layout Suite EXL and higher tiers) Specifies if the first value is 1, the corresponding extension set only applies if the span length of the wire containing the cut is greater than or equal to the second value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

```
'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- 'outsideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Examples

- [Example 1: minQuadrupleExtension with horizontal](#)
- [Example 2: minQuadrupleExtension with allSides and minSum](#)
- [Example 3: minQuadrupleExtension with otherLayer](#)
- [Example 4: minQuadrupleExtension with exactZero](#)
- [Example 5: minQuadrupleExtension with maxLength](#)
- [Example 6: minQuadrupleExtension with allSides and withinFirstWidth](#)
- [Example 7: minQuadrupleExtension with trimLayer and trimLengthTable](#)

Virtuoso Technology Data Constraints Reference

Extension Constraints

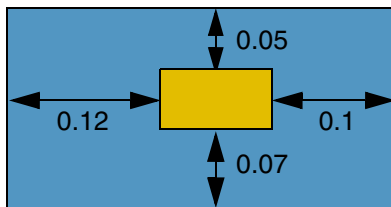
- [Example 8: minQuadrupleExtension with includeAbuttedTable](#)
- [Example 9: minQuadrupleExtension with otherWithinWidthTable](#)
- [Example 10: minQuadrupleExtension in an OR group](#)
- [Example 11: minQuadrupleExtension with metal versus non-via layers](#)
- [Example 12: minQuadrupleExtension with lineEndGapOnlyTable](#)

Example 1: minQuadrupleExtension with horizontal

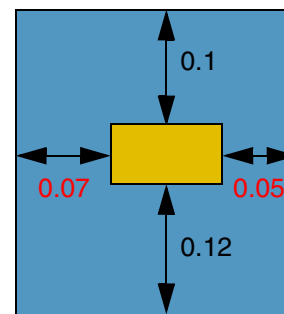
The extensions must be 0.1 and 0.12, measured in the horizontal direction, and 0.05 and 0.07, measured in the vertical direction.

```
spacingTables (  
  ( minQuadrupleExtension "Metal1" "Via1"  
    (( "width" nil nil )  
      'horizontal  
    )  
    ( 0 ((0.1 0.12 0.05 0.07)) )  
  )  
) ;spacingTables
```

■ Via1
■ Metal1



a) PASS. The extensions are 0.12 and 0.1 in the horizontal direction and 0.05 and 0.07 in the vertical direction.



b) FAIL. The extensions in the horizontal direction must be 0.1 and 0.12. The constraint would pass if the direction was not specified.

Virtuoso Technology Data Constraints Reference

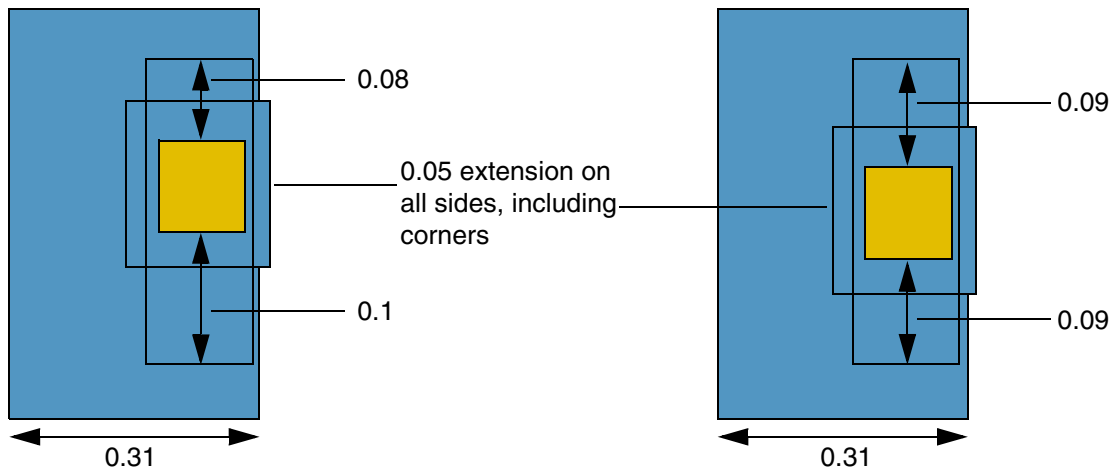
Extension Constraints

Example 2: *minQuadrupleExtension* with *allSides* and *minSum*

A Metal1 shape with width greater than or equal to 0.3 must have a minimum extension of 0.05 on all four sides, including the corners. The extensions of two opposite sides must either be 0.08 and 0.1, or their sum must be greater than or equal to 0.18 and the smaller of the two extensions must be greater than or equal to 0.08.

```
spacingTables(  
  ( minQuadrupleExtension "Metal1" "Via1"  
    (( "width" nil nil )  
      'allSides  
    )  
    ( 0.3 ((0.05 0.05 0.08 0.1 'minSum)) )  
  )  
) ;spacingTables
```

■ Via1
■ Metal1



a) PASS. Metal1 wire with width 0.31 (>0.3) with asymmetric extensions has 0.05 extension on all four sides, including corners. The top and bottom edges satisfy the extension values of 0.08 and 0.1.

b) PASS. The sum of the top and bottom extensions ($0.09+0.09$) is equal to 0.18 ($0.1+0.08$), and the smallest extension of 0.09 is greater than the smaller of the specified extension values (0.08 and 0.1). The constraint would fail if 'minSum' was not specified.

Virtuoso Technology Data Constraints Reference

Extension Constraints

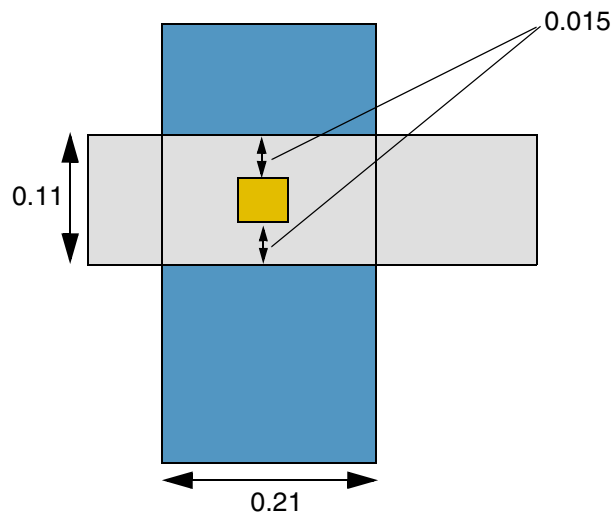
Example 3: *minQuadrupleExtension with otherLayer*

Via1 via cuts must have Metal2 extensions of at least 0.015 on all four sides when the Metal2 wire is greater than or equal to 0.1 wide.

Via1 via cuts must have Metal2 extensions of at least 0.019 on all four sides when the cut overlaps or touches a Metal1 wire that is greater than or equal to 0.2 wide.

```
spacingTables (
  ( minQuadrupleExtension "Metal2" "Via1"
    (( "width" nil nil ))
    ( 0.1 ((0.015 0.015 0.015 0.015)) )
  )
  ( minQuadrupleExtension "Metal2" "Via1"
    (( "width" nil nil ))
    'otherLayer "Metal1"
  )
  ( 0.2 ((0.019 0.019 0.019 0.019)) )
) ;spacingTables
```

□ Metal2
■ Via1
■ Metal1



FAIL. The width of Metal2 is 0.11 (>0.1) and the Via1 cut has a Metal2 extension of 0.015 on all four sides. However, the Via1 cut must have a Metal2 extension of 0.019 (>0.015) on all four sides when it touches or overlaps a Metal1 wire that is 0.21 (>0.20) wide.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 4: *minQuadrupleExtension* with *exactZero*

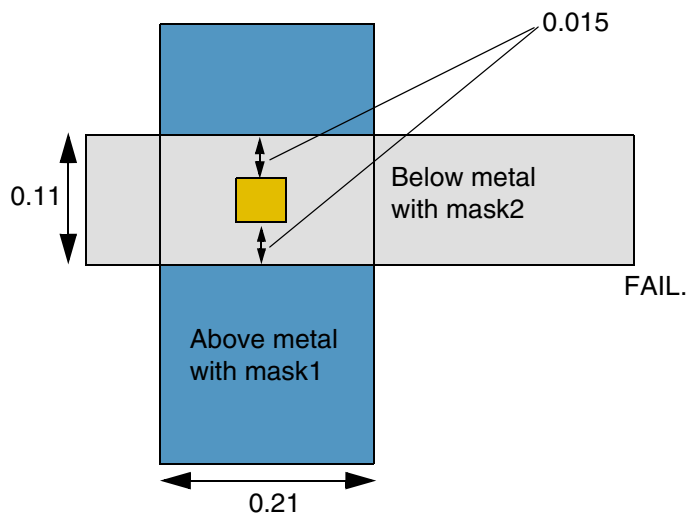
```
spacingTables (
  ;( constraint layer1 [layer2]
  ;(( index1Definitions [index2Definitions]) [defaultValue] )
  ;( table) )
  ;( -----)

  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    'otherLayer "Metal2" 'otherMask1)
    (
      0.2 ((0.019 0.019 0.019 0.019))
    )
  )
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    'exactZeroTable ((("width"))
      (
        0.2 ((1.0 0.0) (0.0 0.0))
        0.3 ((0.0 0.0))
      ))
    )
    (
      0.2 ((0.0 0.0 0.0 0.015) (0.05 0.05 0.018 0.018))
      0.3 ((0.018 0.018 0.02 0.022))
    )
  )
) ;spacingTables
```

□ Metal2

■ Via1

■ Metal1



Virtuoso Technology Data Constraints Reference

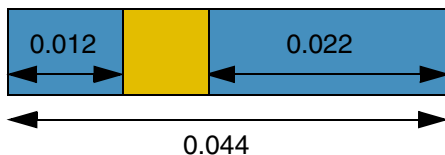
Extension Constraints

Example 5: *minQuadrupleExtension* with *maxLength*

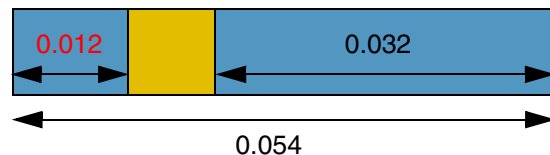
A Metal1 wire with length less than or equal to 0.05 must have a minimum extension of 0.012 on a pair of opposite edges of a Via1 via cut. If the length of the wire is greater than 0.05, the extension must be at least 0.015.

```
spacingTables (
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    ( 0.0 ((0.0 0.0 0.015 0.015)) )
  )
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    ( 0.0 ((0.0 0.0 0.012 0.012 'maxLength 0.05)) )
  )
) ;spacingTables
```

■ Via1
■ Metal1



PASS. The length is 0.044 (<0.05) and the smaller of the two extensions is 0.012.



FAIL. The length is 0.054 (>0.05). Therefore, a minimum extension of 0.015 is required.

Virtuoso Technology Data Constraints Reference

Extension Constraints

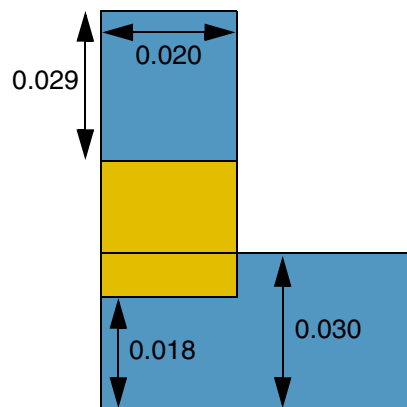
Example 6: minQuadrupleExtension with allSides and withinFirstWidth

A Metal1 wire must have an extension of (0.020 0.015) on two opposite sides of a Via1 via cut that is completely within the wire.

If the via cut is not completely within the wire with width 0.020 and the width of the Metal1 wire that it overlaps is greater than or equal to 0.024, the wire must have a minimum extension of 0.001 on all four sides of the via cut, including the corners. Additionally, the extensions on two opposite sides of the via cut must be (0.020 0.015).

```
spacingTables (
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil )
      'allSides 'withinFirstWidth
    )
    ( 0.000 ((0.020 0.015 0.000 0.000))
      0.024 ((0.020 0.015 0.001 0.001))
    )
  )
) ;spacingTables
```

■ Via1
■ Metal1



FAIL. The enclosure (0.020 0.015 0.000 0.000) required by the first width value specified in the table applies only if the via cut is completely within the wire with width 0.020 (>0.000), a condition that is not satisfied. The enclosure required by the wire with width 0.030 (>0.024) is also not satisfied. The constraint would be met if 'withinFirstWidth' was not specified.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 7: *minQuadrupleExtension* with *trimLayer* and *trimLengthTable*

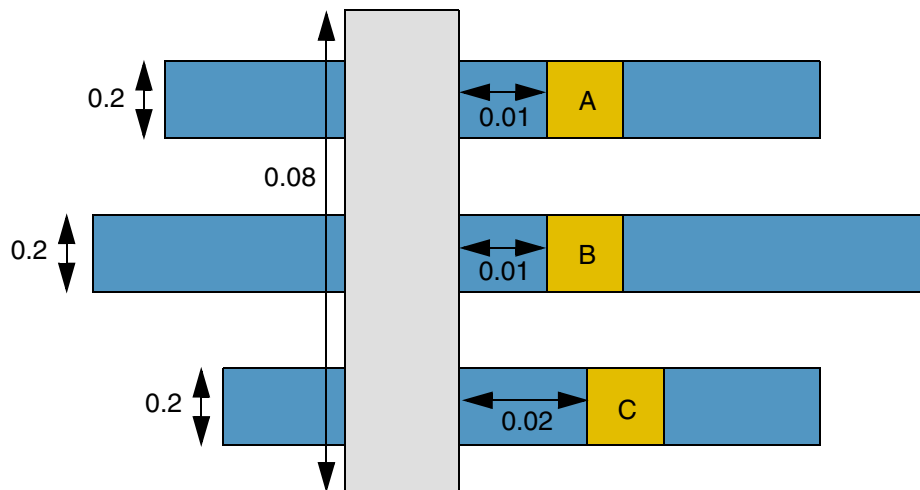
A Metal1 wire, which is overlapped by a TrimMetal1 shape with length greater than or equal to 0.08, must satisfy a minimum extension requirement of 0.02 on two opposite sides of a Via1 via shape. The check applies only to the two outermost wires in the given set.

```
spacingTables (  
  ( minQuadrupleExtension "Metal1" "Via1"  
    (( "width" nil nil )  
      'trimLayer "TrimMetal1"  
      'trimLengthTable ((( "width" ))  
        (  
          0.0 ((0.0 0.0) (0.0 0.0) (0.08 0.08) (0.08 0.08))  
        )  
      )  
    )  
  )  
  (  
    0.0 ((0.01 0.01 0.0 0.0) (0.02 0.02 0.0 0.0))  
  )  
) ;spacingTables
```

Via1

Metal1

TrimMetal1



FAIL. The trim shape is 0.08 in length (≥ 0.08), which means that an extension of 0.02 must be satisfied by the outermost wires that it overlaps. However, of these, only the wire with via C satisfies the minimum extension requirement of 0.02. The check is not performed on the wire with via B because it is in the middle.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 8: minQuadrupleExtension with includeAbuttedTable

In this example, if a cut abuts a wire that is 0.2 wide, the extension values (0.0 0.0 0.0 0.15) need to be met.

```
spacingTables (
  ; ( constraint          layer1          [layer2]
  ;   (( index1Definitions [index2Definitions]) [defaultValue] )
  ;   ( table) )
  ; ( -----)
  ( minQuadrupleExtension "Metall" "Vial"
    (( "width" nil nil )
      'includeAbuttedTable ((("width"))
        (
          0.0 ((0.0 0.0) (0.0 0.0))
          0.2 ((1.0 0.0) (0.0 0.0) (0.0 0.0) (0.0 0.0))
          0.3 ((0.0 0.0) (0.0 0.0))
        ))
      )
      (
        0.0 ((0.0 0.0 0.0 0.013))
        0.2 ((0.0 0.0 0.0 0.015) (0.05 0.05 0.018 0.018))
        0.3 ((0.018 0.018 0.02 0.022))
      )
    )
  ) ;spacingTables
```

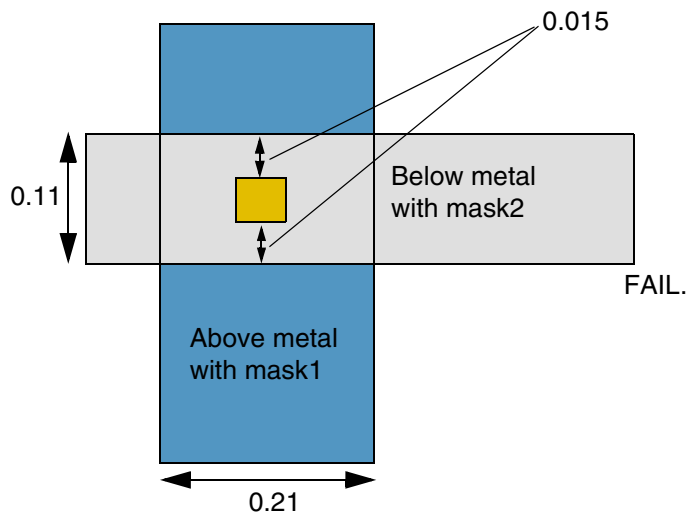
Virtuoso Technology Data Constraints Reference

Extension Constraints

```
spacingTables(
  ;( constraint layer1 [layer2]
  ;(( index1Definitions [index2Definitions]) [defaultValue] )
  ;( table) )
  ;( -----)

  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    'otherLayer "Metal2" 'otherMask1
    (
      0.2 ((0.019 0.019 0.019 0.019))
    )
  )
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    'exactZeroTable ((("width"))
    (
      0.2 ((1.0 0.0) (0.0 0.0))
      0.3 ((0.0 0.0))
    ))
  )
  (
    0.2 ((0.0 0.0 0.0 0.015) (0.05 0.05 0.018 0.018))
    0.3 ((0.018 0.018 0.02 0.022))
  )
) ;spacingTables
```

Metal2
 Via1
 Metal1



Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 9: minQuadrupleExtension with otherWithinWidthTable

For width less than or equal to 0.2, the extension value (0.0 0.0 0.015 0.015) applies only if there is a neighbor wire of width more than 0.05 within 0.07 measured in the horizontal direction. In addition, the extension in the horizontal distance opposite the neighbor wire needs to be at least 0.018.

```
spacingTables (
  ; ( constraint          layer1          [layer2]
    ;   (( index1Definitions [index2Definitions]) [defaultValue] )
    ;   ( table) )
  ; ( -----)
  ( minQuadrupleExtension "Metall" "Vial"
    (( "width" nil nil )
      'otherWithinWidthTable ((("width"))
        (
          0.0 ((0.0 0.0) (0.0 0.0))
          0.2 ((0.05 0.07) (0.0 0.0) (0.0 0.0) (0.0 0.0))
          0.3 ((0.0 0.0) (0.0 0.0))
        ))
      'otherWithinWidthHorizontal
      'otherSideTable ((("width"))
        (
          0.0 ((0.0 0.0) (0.0 0.0))
          0.2 ((0.018 0.0) (0.0 0.0) (0.0 0.0) (0.0 0.0))
          0.3 ((0.0 0.0) (0.0 0.0))
        ))
    )
    (
      0.0 ((0.0 0.0 0.0 0.013))
      0.2 ((0.0 0.0 0.015 0.015) (0.05 0.05 0.018 0.018))
      0.3 ((0.018 0.018 0.02 0.022))
    )
  )
) ;spacingTables
```

Example 10: minQuadrupleExtension in an OR group

The `minExtensionDistance`, `minOppExtension`, and `minQuadrupleExtension` constraints are semantically connected and can be grouped in an OR constraint group. As a result, only one constraint in the constraint group needs to be satisfied, as illustrated in this example.

`minExtensionDistance`

Value = 0.2

Spacing Direction = `horizontal`

`minOppExtension`

Value = (0.15 0.1)

Spacing Direction = `horizontal`

`minQuadrupleExtension`

Width = 0 → ((0.12 0.1) (0.15 0.15))

Spacing Direction = `horizontal`

Virtuoso Technology Data Constraints Reference

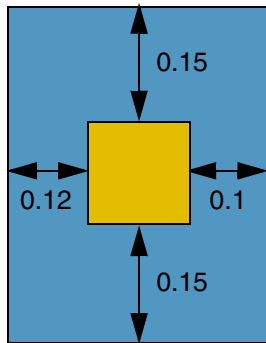
Extension Constraints

```

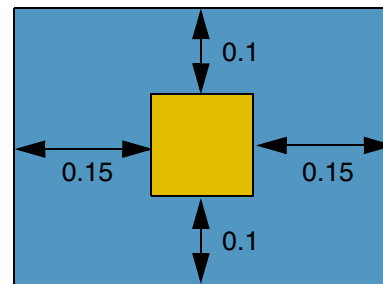
constraintGroups (
  ( "extensionGroup" nil nil 'or
    orderedSpacings (
      ( minExtensionDistance "Metal1" "Via1" 'horizontal 0.2 )
      ( minOppExtension "Metal1" "Via1" 'horizontal (0.15 0.1) )
    ) ;orderedSpacings
    spacingTables (
      ( minQuadrupleExtension "Metal1" "Via1"
        (( "width" nil nil )
          'horizontal
        )
        ( 0 ((0.12 0.1 0.15 0.15)) )
      )
    ) ;spacingTables
  ) ;extensionGroup
  ( "foundry" nil
    memberConstraintGroups ("extensionGroup")
  ) ;foundry
) ;constraintGroups

```

Via1
 Metal1



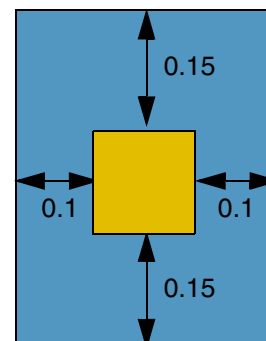
a) PASS. The minQuadrupleExtension constraint is met.



b) PASS. The minOppExtension constraint is met.



c) PASS. The minExtensionDistance constraint is met.



d) FAIL. None of the constraints is met.

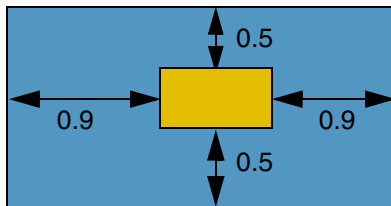
Example 11: minQuadrupleExtension with metal versus non-via layers

This example illustrates the difference between how the constraint applies differently depending on the nature of the two layers.

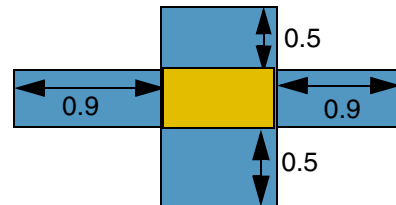
a. Enclosures are used when the constraint is applied between metal and cut layers. The layer2 shape must be fully enclosed on all four sides.

```
spacingTables (
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil )
      )
    ( 0 (0.5 0.5 0.9 0.9) )
  )
) ;spacingTables
```

■ Via1
■ Metal1



i) PASS.

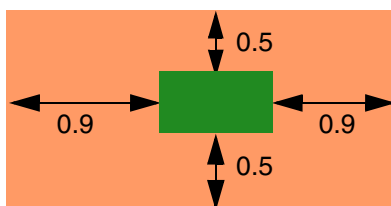


ii) FAIL.

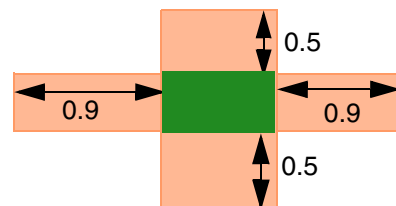
b. Extensions are used when the constraint is applied between device layers or metal over non-via extension. Only the side edges of the layer2 shape must be enclosed.

```
spacingTables (
  ( minQuadrupleExtension "Poly" "Active"
    (( "width" nil nil )
      )
    ( 0 (0.5 0.5 0.9 0.9) )
  )
) ;spacingTables
```

■ Poly
■ Active



i) PASS.



ii) PASS.

Example 12: minQuadrupleExtension with lineEndGapOnlyTable

The first extension set for width 0.2 applies only if the `lineEndGap` condition for Metal1 is met.

```
spacingTables(  
  ...  
  ( minQuadrupleExtension "Metal1" "Via1"  
    (( "width" nil nil )  
      'lineEndGapOnlyTable ((( "width" )  
        (  
          0.0 ((0.0 0.0) (0.0 0.0))  
          0.2 ((1.0 0.0) (0.0 0.0) (0.0 0.0) (0.0 0.0))  
          0.3 ((0.0 0.0) (0.0 0.0))  
        ))  
      )  
      (  
        0.0((0.0 0.0 0.0 0.013))  
        0.2((0.0 0.0 0.015 0.015) (0.05 0.05 0.018 0.018))  
        0.3((0.018 0.018 0.02 0.022))  
      )  
    )  
  )  
;spacingTables
```

minSideExtension

```
orderedSpacings (
    ( minSideExtension tx_layer1 tx_layer2
      ['otherExtension f_otherExtension]
      ['longEdge]
      ['widthRanges (g_ranges) ['otherWidthRanges (g_otherWidthRanges)]]
      ['exceptEdgeLengthRanges (g_lengthRanges)]
      f_extension
    )
) ;orderedSpacings
```

Specifies the minimum extension of a *layer1* shape past a pair of opposite sides of a *layer2* shape.

Optionally, you can specify if the constraint applies only to the long or short edges of the *layer1* shape or to *layer1* and *layer2* shapes with width in the specified ranges. Additionally, shapes on *layer2* with length in the specified ranges can be exempted.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The extension of a <i>layer1</i> shape past a pair of opposite sides of a <i>layer2</i> shape must be greater than or equal to this value.

Parameters

'otherExtension <i>f_otherExtension</i>	The constraint applies only if the extension of the <i>layer2</i> shape past the <i>layer1</i> shape is less than or equal to this value.
'longEdge	The constraint applies to the long edge of the <i>layer1</i> shape; otherwise, the constraint applies to the short edge. Type: Boolean

Virtuoso Technology Data Constraints Reference

Extension Constraints

`'widthRanges g_ranges`

The constraint applies only if the width of the *layer1* shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint

`'otherWidthRanges g_otherWidthRanges`

The constraint applies only if the width of the *layer2* shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint

`'exceptEdgeLengthRanges g_lengthRanges`

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if the length of the *layer2* shape falls in this range.

Type: Floating-point values specifying a range of lengths that are exempted

Examples

- Example 1: minSideExtension
- Example 2: minSideExtension with otherExtension
- Example 3: minSideExtension with longEdge and exceptEdgeLengthRanges

Virtuoso Technology Data Constraints Reference

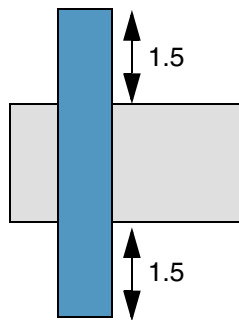
Extension Constraints

Example 1: *minSideExtension*

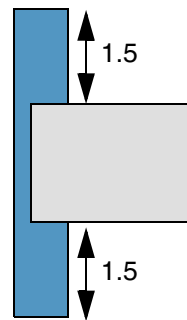
The extension of a Metal1 shape on two opposite sides of a Metal2 shape must be at least 1.5.

```
orderedSpacings (  
  ( minSideExtension "Metal1" "Metal2"  
    1.5  
  )  
) ;orderedSpacings
```

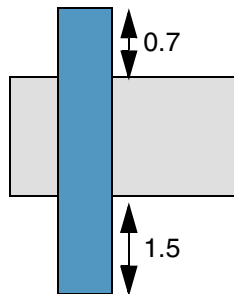
□ Metal2
■ Metal1



a) PASS. The extension of the Metal1 shape on two opposite sides of the Metal2 shape is 1.5.



a) PASS. The extension of the Metal1 shape on two opposite sides of the Metal2 shape is 1.5.



c) FAIL. The extension of the Metal1 shape past the Metal2 shape is 1.5 on only one side.

Virtuoso Technology Data Constraints Reference

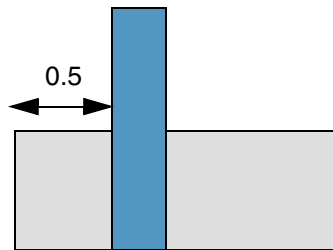
Extension Constraints

Example 2: *minSideExtension* with *otherExtension*

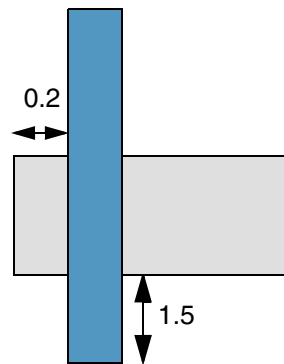
The extension of a Metal1 shape on a pair of opposite sides of a Metal2 shape must be at least 2.0 if the extension of one side of the Metal2 shape past the Metal1 shape is less than or equal to 0.2.

```
orderedSpacings (  
  ( minSideExtension "Metal1" "Metal2"  
    'otherExtension 0.2  
    2.0  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



a) The constraint does not apply because the extension of the Metal2 shape past the Metal1 shape is 0.5 (>0.2).



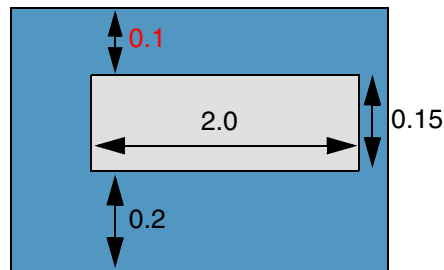
a) FAIL. The extension of the Metal2 shape past a side of the Metal1 shape is 0.2, but the extension of the Metal1 shape past the Metal2 shape is only 1.5 (<2.0).

Example 3: minSideExtension with longEdge and exceptEdgeLengthRanges

The extension of a long edge of a Metal1 shape past a Metal2 shape must be greater than or equal to 0.12, except when the Metal2 edge is less than or equal to 0.2, equal to 0.5, or equal to 0.6.

```
orderedSpacings (
  ( minSideExtension "Metal1" "Metal2"
    'longEdge
    'exceptEdgeLengthRanges ("<=0.2" "0.5" "0.6")
    0.12
  )
) ;orderedSpacings
```

□ Metal2
■ Metal1



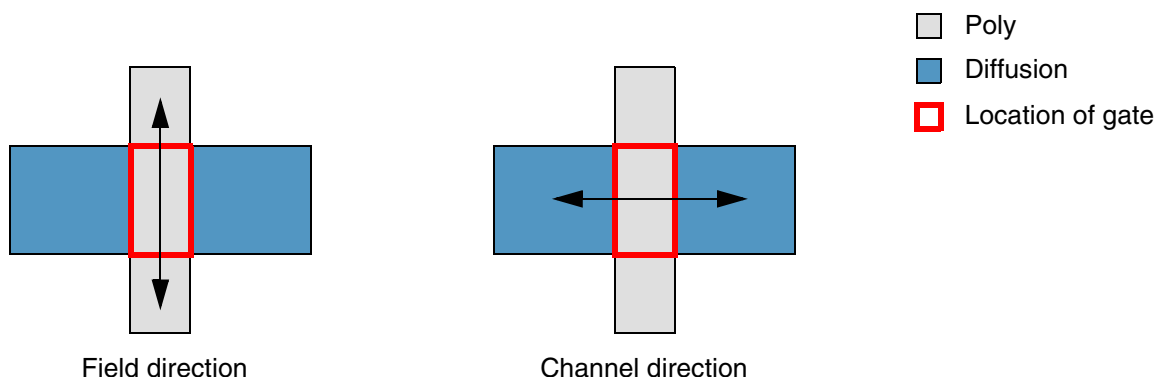
FAIL. The constraint applies because the length of the Metal2 shape does not fall in the exempted length ranges (but applies only to the top and bottom Metal1 edges because 'longEdge is specified). However, the extension of the top edge of the Metal1 shape past the Metal2 shape is only 0.1 (<0.12).

minTouchingDirEnclosure

```
orderedSpacings (  
    ( minTouchingDirEnclosure tx_layer1 tx_layer2 tx_layer3  
      f_enclosure ['manhattan']  
    )  
);orderedSpacings
```

Specifies the minimum enclosure for the field end of the gate (this is different from the enclosure for the channel).

The direction of field-end enclosure is based on where the top and bottom edges of the gate touch "Poly", and the direction of the channel enclosure is based on where the sides of the gate touch "Diffusion", as shown below.

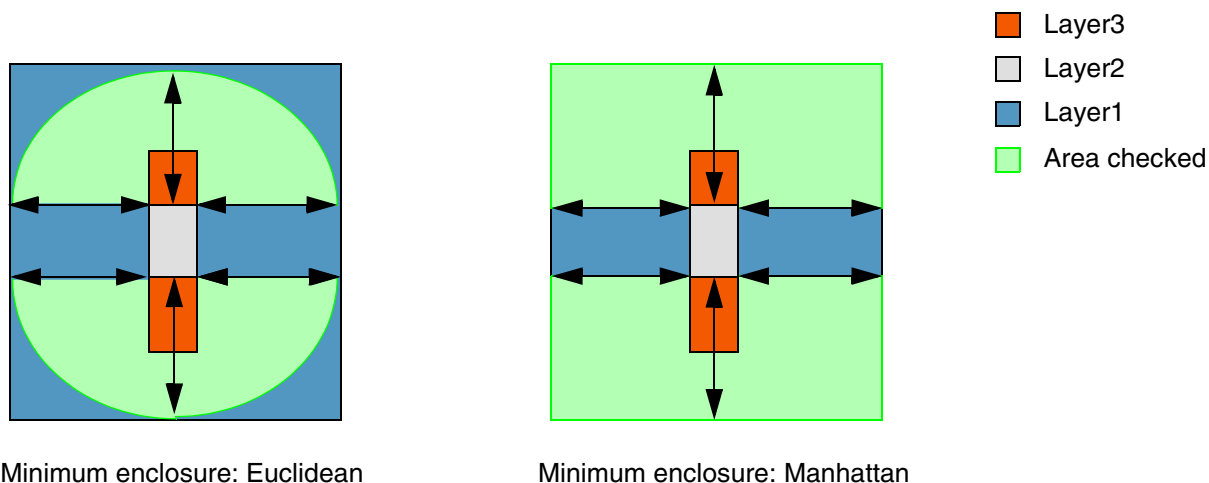


Note: Usually, the `minTouchingDirEnclosure` constraint is used to check the overlap of "Poly" over "Diffusion".

Virtuoso Technology Data Constraints Reference

Extension Constraints

The enclosure direction is at 90 degrees to the perimeter edges of the *layer2* shape that touch the *layer3* (the edges are either enclosed by or abut the *layer3* shape), where the *layer3* shape extends beyond the *layer2* shape in the same direction, as shown below:



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. The shape on this layer encloses a shape on <i>tx_layer2</i>. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. The shape on this layer is enclosed. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer3</i>	The third layer on which the constraint is applied. The shape on this layer is the reference shape whose placement in relation to the <i>layer2</i> shape determines how the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_enclosure</i>	The enclosure of the field end of the gate must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

'manhattan	The constraint uses Manhattan distance, which allows a larger spacing at the corners. By default, the constraint uses Euclidean measurement. Type: Boolean

Example

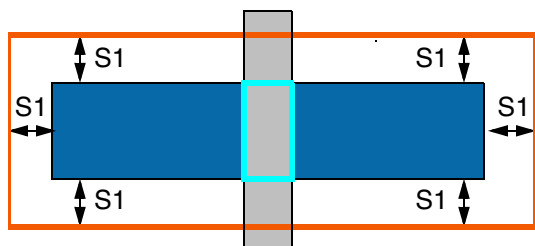
The `minEnclosure` and `minTouchingDirEnclosure` constraints are used with a derived layer named `polyAndDiffusion` (equivalent to `layer3`) to create a larger minimum implant enclosure around the `poly` endcap, than is applied around the rest of the diffusion.

```
layerDefinitions (
  ( techDerivedLayers((polyAndDiffusion 1000 (poly 'and diffusion))) )
) ;layerDefinitions
orderedSpacings (
  ( minTouchingDirEnclosure "implant" "polyAndDiffusion" "poly"
    2.0 'manhattan
  )
  ( minEnclosure "implant" "diffusion"
    1.0
  )
) ;orderedSpacings
```

Virtuoso Technology Data Constraints Reference

Extension Constraints

Minimum enclosure of diffusion by implant specified by constraint `minEnclosure layer1 (implant) enclosing layer2 (diffusion) = 1.0`



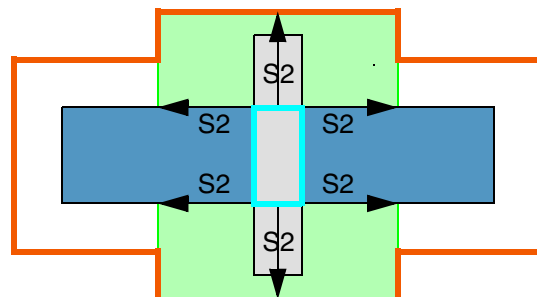
- poly
- diffusion
- polyAndDiffusion
- implant
- Area checked

Combining `minTouchingDirEnclosure` with `minEnclosure` can create a different enclosure size for the poly endcap by the implant layer.

<code>minTouchingDirEnclosure = 2.0 (S2)</code>	<code>minEnclosure = 1.0 (S1)</code>
<code>layer1 = implant</code>	<code>layer1 = implant</code>
<code>layer2 = polyAndDiffusion</code>	<code>layer2 = diffusion</code>
<code>layer3 = poly</code>	

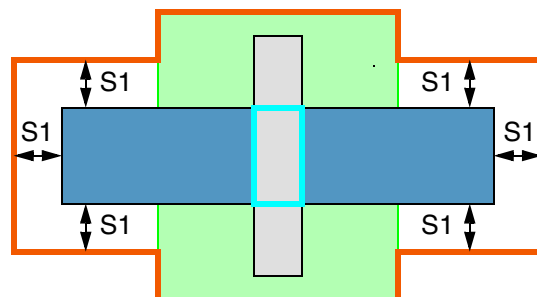
`minTouchingDirEnclosure` applies:

- Perpendicular to the touching edges in the direction in which `layer3` extends beyond `layer2`
- From the corners of the touching edges, perpendicular to the direction in which `layer3` extends beyond `layer2`



AND

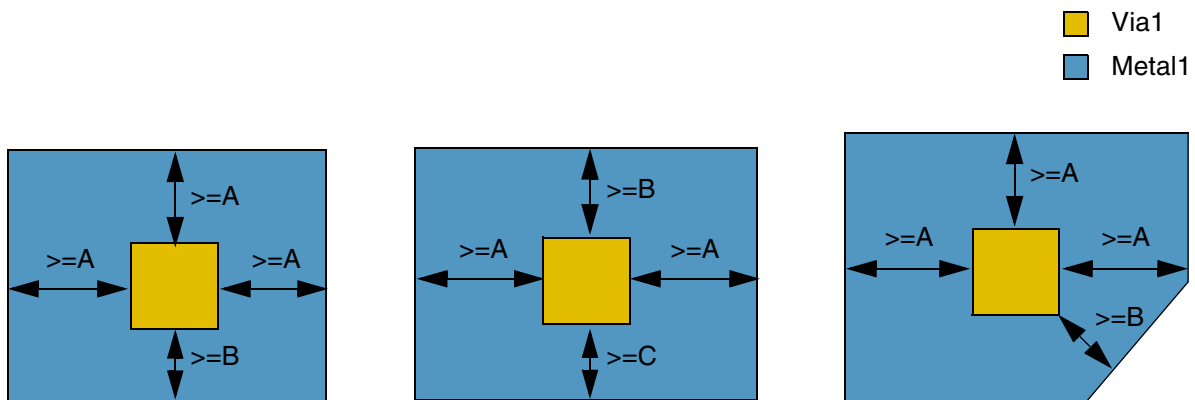
`minEnclosure` of `layer1` enclosing `layer2` applies to the remaining areas of implant over diffusion



minViaExtension

```
orderedSpacings (
  ( minViaExtension tx_layer1 tx_layer2
    'oneThree (f_ext1 f_ext2)
    | 'oneOneTwo (f_ext1 f_ext2 g_ext3)
    | 'diagonal (f_ext1 f_ext2 [f_ext3 [f_ext4]])
  )
) ;orderedSpacings
```

Specifies the minimum extension of a *layer1* shape past a *layer2* shape, where extensions can be different on different sides depending on the number of extension values that are specified, as shown below.



Values

tx_layer1

The metal layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

tx_layer2

The cut layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

`'oneThree (f_ext1 f_ext2)`

The first extension value, *ext1*, applies to any three sides, and the second extension value, *ext2*, applies to the fourth side.

`'oneOneTwo (f_ext1 f_ext2 g_ext3)`

The first extension value, *ext1*, applies to two opposite sides, the second extension value, *ext2*, applies to one of the remaining two sides, and the third extension value, *ext3*, applies to the last remaining side.

`'diagonal (f_ext1 f_ext2 [f_ext3 [f_ext4]])`

The following three scenarios apply:

- If two extension values are specified, the first one is used for non-diagonal edges and the second one is for diagonal edges.
- If three extension values are specified, the first two work as with `'oneThree`, and the third value is the required extension of a diagonal edge.
- If four extension values are specified, the first three work as with `'oneOneTwo`, and the fourth is the required extension to a diagonal edge.

Example

A Metal1 shape must have the following extensions on the sides of a Via1 via cut:

- 0.2 on two opposite sides
- 0.3 on one of the two remaining sides
- 0.4 on the fourth side

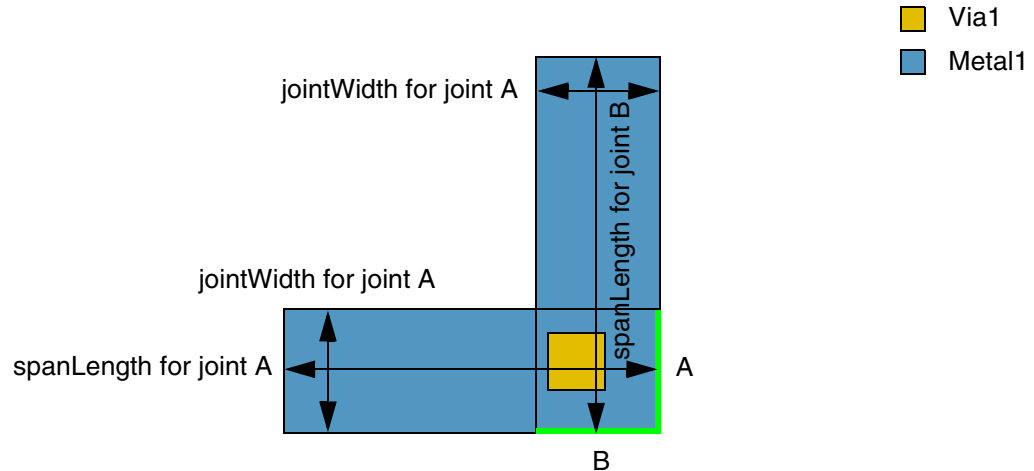
```
orderedSpacings (  
  ( minViaExtension "Metal1" "Via1" 'oneOneTwo (0.2 0.3 0.4) )  
) ;orderedSpacings
```


minViaJointExtension

```
orderedSpacings (
  ( minViaJointExtension tx_layer1 tx_layer2
    ['cutClass {f_width | (f_width f_length) | t_name}]
    'jointWidth f_jointWidth
    'spanLength f_spanLength
    ['eolMinLength f_eolMinLength]
    f_extension | (f_extension1 f_extension2)
  )
) ;orderedSpacings
```

Specifies the minimum extension of a *layer1* shape past the joint edges of a *layer2* shape of the specified cut class.

In some advanced node processes, cut shapes require different extensions on joint edges as compared to regular edges. The following figure illustrates joint edges (in green):



Virtuoso Technology Data Constraints Reference

Extension Constraints

Values

<code>tx_layer1</code>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>tx_layer2</code>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>f_extension (f_extension1 f_extension2)</code>	The extension of one of the joint edges must be greater than or equal to <i>extension</i> ; the extension of one of the joint edges must be greater than or equal to <i>extension1</i> or the extension of both joint edges must be must be greater or equal to <i>extension2</i> .

Parameters

<code>'cutClass {f_width (f_width f_length) t_name}</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <code>f_width</code>: Width ■ <code>f_length</code>: Length ■ <code>t_name</code>: Name of the cut class
<code>'jointWidth f_jointWidth</code>	The constraint applies only if the width of a joint is less than this value.
<code>'spanLength f_spanLength</code>	The constraint applies only if the span length of a joint is greater than this value.
<code>'eolMinLength f_eolMinLength</code>	The joint must not be an end-of-line edge with both adjoining edges greater than or equal to this value in length.

Virtuoso Technology Data Constraints Reference

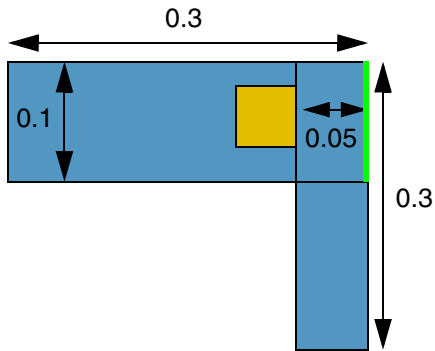
Extension Constraints

Example

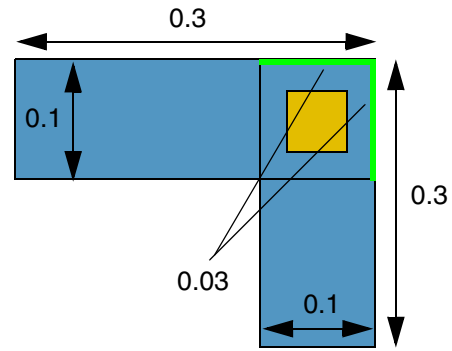
If joint width is less than 0.15 and span length is greater than 0.1, the extension of one of the joint edges (Metal1) must be greater than or equal to 0.05 or the extension of both joint edges must be greater than or equal to 0.03.

```
orderedSpacings (  
  ( minViaJointExtension "Metal1" "Via1"  
    'jointWidth 0.15  
    'spanLength 0.1  
    (0.05 0.03)  
  )  
); orderedSpacings
```

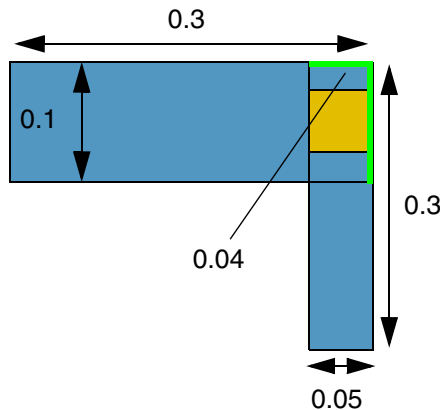
■ Via1
■ Metal1



a) PASS. The width of the joint is 0.1 (<0.15) and the span length is 0.3 (>0.1); the extension of the joint edge is 0.05.



b) PASS. The width of both joints is 0.1 (<0.15) and both span lengths are 0.3 (>0.1); the extension of both joint edges is 0.03.



c) FAIL. The width of both joints is less than 0.15 and both span lengths are 0.3 (>0.1), but the extensions of the joint edges are 0 and 0.04, which does not satisfy the extension requirement.

minVoltageExtension

```
spacingTables(  
  ( minVoltageExtension tx_layer1 tx_layer2  
    (( "voltage" nil nil )  
      ['horizontal | 'vertical]  
    )  
    (g_table)  
  )  
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Only) Specifies the minimum extension of a *layer1* shape past a *layer2* shape when the voltage on *layer1* is greater than or equal to the specified value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Values

<code>tx_layer1</code>	The first layer on which the constraint is applied. The shape on this layer encloses a shape on <code>tx_layer2</code>. Type: String (layer and purpose names) or Integer (layer number)
<code>tx_layer2</code>	The second layer on which the constraint is applied. The shape on this layer is enclosed. Type: String (layer and purpose names) or Integer (layer number)
<code>"voltage" nil nil</code>	This identifies the index for <code>table</code>.
<code>g_table</code>	The format of the <code>table</code> row is as follows: <code>(f_voltage f_extension)</code> where, <code>f_extension</code> is the extension of the <code>layer1</code> shape past the <code>layer2</code> shape. The <code>layer1</code> extension on all four sides must be greater than or equal to <code>f_extension</code> if the voltage on <code>layer1</code> is greater than or equal to <code>f_voltage</code>. Type: A 1-D table specifying floating-point extension and voltage values

Parameters

<code>'horizontal 'vertical</code>	The constraint applies only to the extensions measured in this direction. If direction is not specified, the extension is measured in any direction. Type: Boolean

Examples

- [Example 1: minVoltageExtension with voltage](#)
- [Example 2: minVoltageExtension with direction](#)

Virtuoso Technology Data Constraints Reference

Extension Constraints

Example 1: minVoltageExtension with voltage

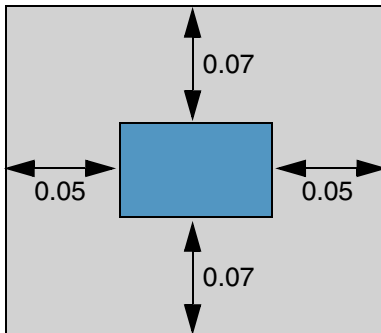
The extension of an Nwell shape past a Pdiff shape must be as follows:

- At least 0.05 if the voltage on the Nwell shape is greater than or equal to 1.0 and less than or equal to 1.5
- At least 0.07 if the voltage on the Nwell shape is greater than or equal to 1.5

```
spacingTables (
  ( minVoltageExtension "Nwell" "Pdiff"
    ( "voltage" nil nil )
    (
      1.0 0.05
      1.5 0.07
    )
  )
) ;spacingTables
```

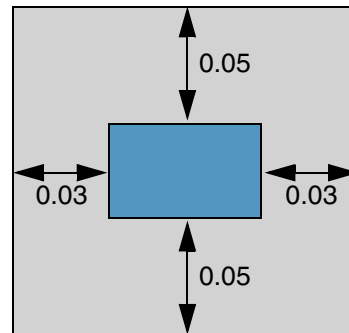
□ Nwell
■ Pdiff

Nwell voltage: ≥ 1.0 to ≤ 1.2



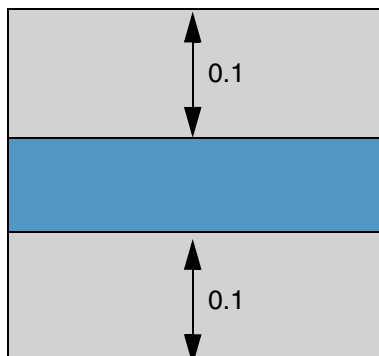
a) PASS. The voltage on Nwell is greater than or equal to 1.0 and the extensions on all sides are greater than or equal to 0.05.

Nwell voltage: ≥ 1.0 to ≤ 1.2



b) FAIL. The voltage on Nwell is greater than or equal to 1.0, but the extensions on the left and right sides are only 0.03 (< 0.05).

Nwell voltage: ≥ 1.5



c) FAIL. The voltage on Nwell is greater than or equal to 1.5, but the extensions on the left and right sides are 0 (< 0.07).

Virtuoso Technology Data Constraints Reference

Extension Constraints

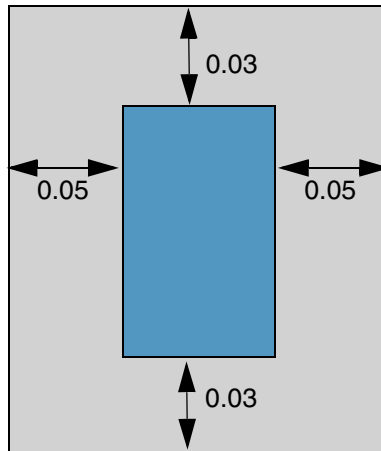
Example 2: minVoltageExtension with direction

This example illustrates the use of the direction parameter.

```
spacingTables (
  ( minVoltageExtension "Nwell" "Pdiff"
    ( "voltage" nil nil )
    'horizontal
    (
      1.0 0.05
      1.5 0.07
    )
  )
) ;spacingTables
```

□ Nwell
■ Pdiff

Nwell voltage: ≥ 1.0 to ≤ 1.2



PASS. The voltage on Nwell is greater than or equal to 1, and the extension of Nwell over pdiff is greater than or equal to 0.05 in the horizontal direction

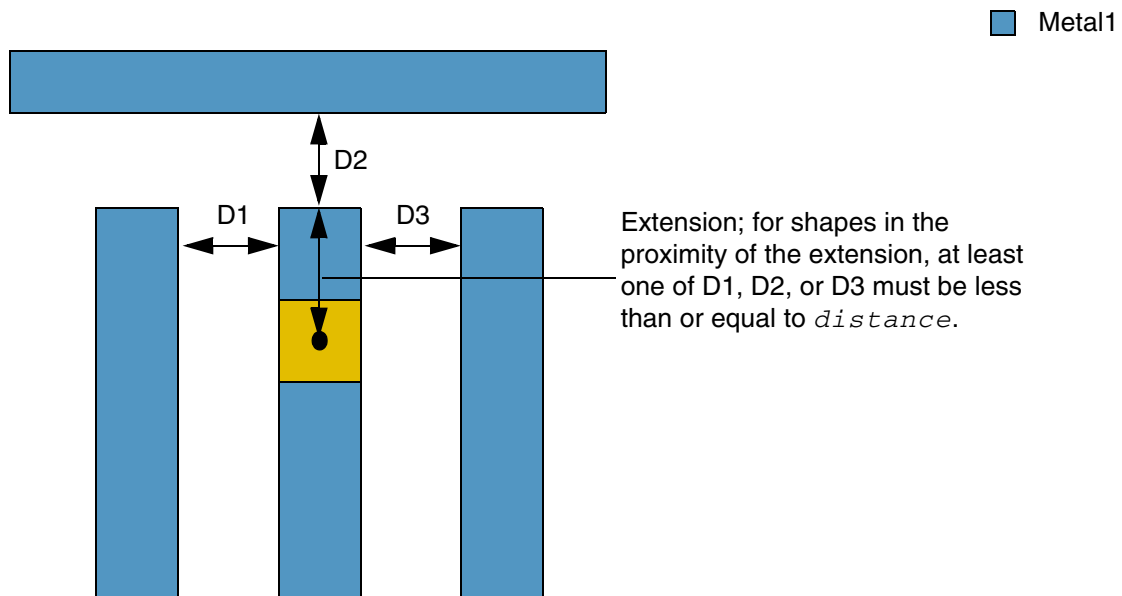
If the direction was vertical, this would be FAIL because the voltage on Nwell is greater than or equal to 1, and the extension of Nwell over pdiff is 0.03 (which is less than 0.05) in the vertical direction.

If the direction was not specified, this would be FAIL because the voltage on Nwell is greater than or equal to 1, and the extension of Nwell over pdiff is 0.03 (which is less than 0.05) in the vertical direction.

minWireExtension

```
spacings (
    ( minWireExtension tx_layer
      {f_extension | (f_distance f_extension)}
      ['coincidentAllowed']
    )
) ; spacings
```

Specifies the minimum extension of a wire on *layer* past the center of a via cut that can be on a cut layer above or below *layer*.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	The distance from the center of the via cut to the edge of the enclosing wire must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Extension Constraints

Parameters

f_distance

The constraint applies only if the distance between the extension and a shape in the proximity of the extension on the same layer is less than or equal to this value.

'coincidentAllowed

The edges of the via cut can coincide with the edges of the enclosing wire. Otherwise, the extension requirement must be met.

Type: Boolean

Example

The distance a Metal3 wire must extend past the center of a via cut on the cut layer above or below Metal3 must be greater than or equal to 1.5.

```
spacings(  
    ( minWireExtension "Metal3"  
      1.5  
    )  
); spacings
```

Virtuoso Technology Data Constraints Reference

Extension Constraints

LEF Constraints

This topic includes the following LEF legacy compatibility constraints:

- LEF_AREACAPACITANCE
- LEF_CAPMULTIPLIER
- LEF_EDGECAPACITANCE
- LEF_HEIGHT
- LEF_SHRINKAGE

LEF_AREACAPACITANCE

```
spacings (
    ( LEF_AREACAPACITANCE tx_layer
      f_areaCap
    )
) ;spacings
```

Specifies the wire-to-ground capacitance for each square unit, in picofarads per square micron.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_areaCap</i>	The required area capacitance.

Parameters

None.

Example

The capacitance on Metal1 is 0.16.

```
spacings (
    ( LEF_AREACAPACITANCE "Metal1"
      0.16
    )
) ;spacings
```

LEF_CAPMULTIPLIER

```
spacings (
    ( LEF_CAPMULTIPLIER tx_layer
      x_multiplier
    )
) ;spacings
```

Specifies the multiplier for interconnect capacitance to account for increases in capacitance caused by nearby wires.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_multiplier</i>	The required capacitance multiplier.

Parameters

None.

Example

The capacitance multiplier on Metal1 is 2.

```
spacings (
    ( LEF_CAPMULTIPLIER "Metal1"
      2
    )
) ;spacings
```

LEF_EDGECAPACITANCE

```
spacings (
    ( LEF_EDGECAPACITANCE tx_layer
      f_edgeCap
    )
) ;spacings
```

Specifies a floating-point value of peripheral capacitance, in picofarads per micron. The place-and-route tool uses this value in these situations:

- Estimating capacitance before routing.
- Calculating segment capacitance after routing. The tool uses the value only if you set the layer thickness or layer height to 0. The following formula is used:

$(\text{layer capacitance per square} \times \text{segment width} \times \text{segment length}) + (\text{peripheral capacitance} \times 2 \times (\text{segment width} + \text{segment length}))$

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_edgeCap</i>	The required edge capacitance.

Parameters

None.

Example

The edge capacitance on Metal1 is 0.0000883.

```
spacings (
    ( LEF_EDGECAPACITANCE "Metal1"
      0.0000883
    )
) ;spacings
```

LEF_HEIGHT

```
spacings (
    ( LEF_HEIGHT tx_layer
      f_height
    )
) ;spacings
```

Specifies the distance from the top of the ground plane to the bottom of the interconnect.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_height</i>	The required height.

Parameters

None.

Example

The height for Metal1 is 0.59.

```
spacings (
    ( LEF_HEIGHT "Metal1"
      0.59
    )
) ;spacings
```

LEF_SHRINKAGE

```
spacings (
    ( LEF_SHRINKAGE tx_layer
      f_dist
    )
) ;spacings
```

Specifies the value to account for the shrinkage of interconnect wiring due to the etching process. Wire widths are determined by subtracting this constant value.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_dist</i>	The required shrinkage.

Parameters

None.

Example

The shrinkage for Metal1 is 0.0001.

```
spacings (
    ( LEF_SHRINKAGE "Metal1"
      0.0001
    )
) ;spacings
```

Length and Width Constraints

This topic includes the following constraints:

- [allowedGateLengthRanges](#)
- [allowedGateWidthRanges](#)
- [allowedLengthRanges](#)
- [allowedNeighborWidthRanges \(One layer\)](#)
- [allowedNeighborWidthRanges \(Two layers\)](#)
- [allowedNeighborWidthRangesOver](#)
- [allowedPRBoundaryDimensions](#)
- [allowedSpanLengthRanges](#)
- [allowedWidthRanges](#)
- [boundaryEOLBlockage](#)
- [illegalHGatePattern](#)
- [maxDiagonalEdgeLength](#)
- [maxDiffusionLength](#)
- [maxEdgeLength](#)
- [maxLength](#)
- [maxLengthWithCut](#)
- [maxNumMinEdges](#)
- [maxPolyLength](#)
- [maxTouchingDirectionLength](#)
- [maxWidth](#)

Virtuoso Technology Data Constraints Reference

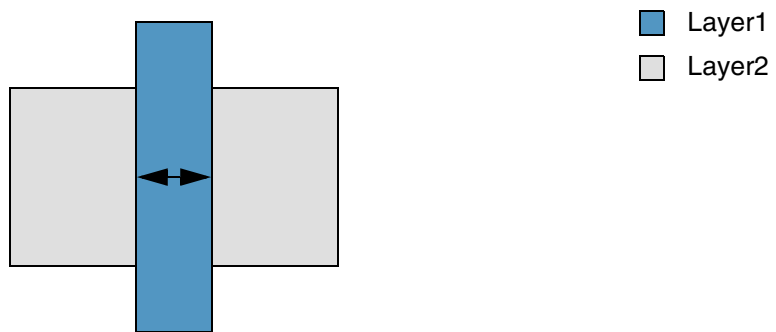
Length and Width Constraints

- minChamferLength
- minDiagonalEdgeLength
- minDiagonalWidth
- minEdgeAdjacentDistance
- minEdgeAdjacentLength
- minEndOfLineAdjacentToStep
- minInsideCornerEdgeLength
- minLength
- minOutsideCornerEdgeLength
- minPerimeter
- minProtrusionWidthLength
- minSize
- minStepEdgeLength
- minWidth
- protrusionWidth

allowedGateLengthRanges

```
orderedSpacings (
    ( allowedGateLengthRanges tx_layer1 tx_layer2
      (g_ranges)
    )
) ;orderedSpacings
```

Specifies the allowed channel length. The channel length is the width of the poly shape over active.



Values

<i>tx_layer1</i>	The first layer (poly) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer (active) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The allowed ranges for the channel length (that is, poly width). Type: Floating-point values specifying length <u>ranges</u>

Parameters

None

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Example

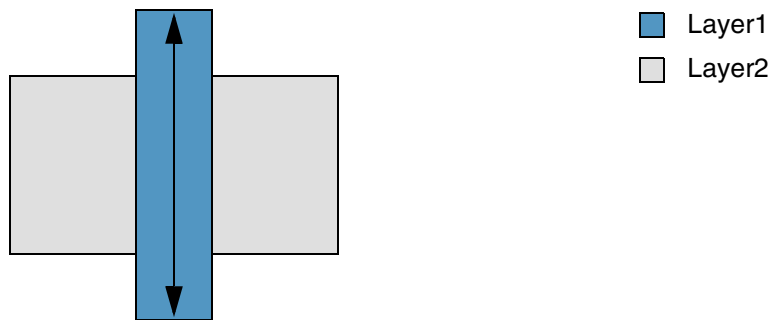
The channel length must be greater than 0.2 and less than 0.4 or greater than or equal to 1.0.

```
orderedSpacings (
  ( allowedGateLengthRanges "Poly" "Active"
    ("(0.2 0.4)" ">=1.0")
  )
) ;orderedSpacings
```

allowedGateWidthRanges

```
orderedSpacings (
  ( allowedGateWidthRanges tx_layer1 tx_layer2
    (g_ranges)
  )
) ;orderedSpacings
```

Specifies the allowed channel width. The channel width is the length of the poly shape over active.



Values

<i>tx_layer1</i>	The first layer (poly) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer (active) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The allowed ranges for the channel width (that is, poly length). Type: Floating-point values specifying width <u>ranges</u>

Parameters

None

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Example

The channel width must be greater than 0.2 and less than 0.4 or greater than or equal to 1.0.

```
orderedSpacings (
  ( allowedGateWidthRanges "Poly" "Active"
    ("(0.2 0.4)" ">= 1")
  )
) ;orderedSpacings
```

allowedJogWidths

```
spacingTables(
  ( allowedJogWidths tx_layer
    (( "width" nil nil )
      {'horizontal | 'vertical}
      ['snapPatternDefs l_spDefNames]
      ['mask1 | 'mask2 | 'mask3]
      ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
          ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      g_rangeTable
    )
  ) ;spacingTables
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the width-to-width transitions allowed from the preferred direction on the specified layer to the non-preferred direction of the same layer and the snapping requirements for the centerline of shapes in the non-preferred direction.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil	This identifies the index for <i>rangeTable</i> .
<i>g_rangeTable</i>	The format of the <i>rangeTable</i> row is as follows: (<i>f_width</i> <i>l_valueRange</i>) where, ■ <i>f_width</i> is the width of the wire measured in the orthogonal direction. ■ <i>l_valueRange</i> is a list of <u>ranges</u> indicating the allowed jog widths for the index as measured in the specified direction.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

'horizontal | 'vertical

The direction in which length is measured.

Type: Boolean

'snapPatternDefs *l_spDefNames*

The list of width spacing snap pattern definitions to which this constraint applies.

The list of snapPatternDefs or widthSpacingSnapPatternDefs. The centerlines of shapes in the non-preferred direction should snap to one of the specified tracks.

'mask1 | 'mask2 | 'mask3

The mask to which the table of transitions of this rule belongs to.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

(Virtuoso Layout Suite EXL and higher tiers) Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Layout Suite EXL and higher tiers) The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

(Virtuoso Layout Suite EXL and higher tiers) The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

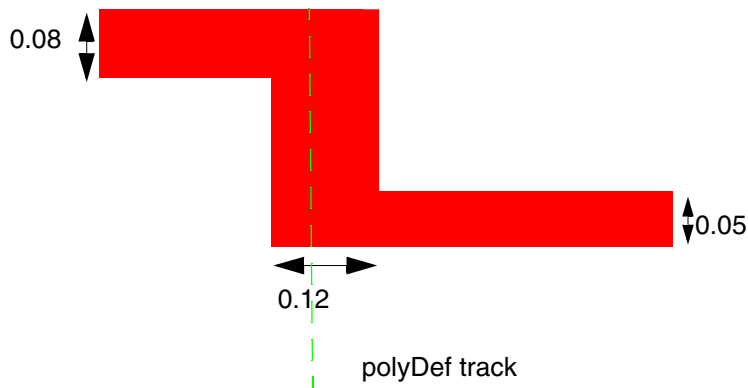
Length and Width Constraints

Example

For horizontal M2 wires of width 0.05 and 0.08, measured vertically, the allowed jog widths in vertical direction are 0.06 and 0.12, measured horizontally. Also, the center lines of vertical wires need to snap to `polyDef` or `m1Def` tracks.

```
(spacingTables
  (allowedJogWidths "m2"
    (( "width" nil nil)
     'snapPatternDefs ("polyDef" "m1Def")
     'horizontal)
    (
      0.05 (0.06 0.12)
      0.08 (0.06 0.12)
    )
  ))
```

■ M2



PASS. The vertical wire has an allowed jog width of 0.12 measured in the horizontal direction and its centerline is snapped to a track of `polyDef`.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

allowedLengthRanges

```
spacings (
  ( allowedLengthRanges tx_layer
    ['horizontal' | 'vertical']
    [{ 'viaLayers (tx_viaLayer1 tx_viaLayer2 ... tx_viaLayerN) |
      'exceptViaLayers (tx_exceptViaLayer1 tx_exceptViaLayer2 ...
      tx_exceptViaLayerN) }
    ['cutClassSizes ({ f_width | (f_width f_length) | t_name } ... )]
    ]
    ['mask1' | 'mask2' | 'mask3']
    ['endToEndGapRanges g_endToEndGapRanges]
    ['otherEndToEndGapRanges g_otherEndToEndGapRanges]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    (g_ranges)
  )
) ;spacings

spacingTables (
  ( allowedLengthRanges tx_layer
    (( "width" nil nil )
    ['horizontal' | 'vertical']
    [{ 'viaLayers (tx_viaLayer1 tx_viaLayer2 ... tx_viaLayerN) |
      'exceptViaLayers (tx_exceptViaLayer1 tx_exceptViaLayer2 ...
      tx_exceptViaLayerN) }
    ['cutClassSizes ({ f_width | (f_width f_length) | t_name } ... )]
    ]
    ['mask1' | 'mask2' | 'mask3']
    ['endToEndGapRanges g_endToEndGapRanges]
    ['otherEndToEndGapRanges g_otherEndToEndGapRanges]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    [f_default]
  )
  (g_table)
)
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) Specifies the allowed length values for shapes on a layer.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Polygon shapes are broken down into rectangles and the longer dimension of each rectangle is checked against the allowed length values.

Note: If the table value is used, the only allowed indexes are the widths specified with the `allowedWidthRanges` constraint.

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>g_ranges</code>	The allowed length ranges. Type: Floating-point values specifying length <u>ranges</u>
<code>"width" nil nil</code>	This identifies the index for <i>table</i> .
<code>g_table</code>	The format of the <i>table</i> row is as follows: <code>(f_width (g_ranges))</code> where, ■ <code>f_width</code> is the width of the shape. ■ <code>g_ranges</code> is a list of the allowed length <u>ranges</u> for the specified width. Type: A 1-D table specifying floating point width and length values

Parameters

<code>'horizontal 'vertical</code>	(Virtuoso Layout Suite EXL and higher tiers) The direction in which length is measured. If not specified, the length is measured in any direction. Type: Boolean
<code>'viaLayers (tx_viaLayer1 tx_viaLayer2 ... tx_viaLayerN)</code>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if a shape is present on any of these layer.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

`'exceptViaLayers (tx_exceptViaLayer1 tx_exceptViaLayer2
...tx_exceptViaLayerN)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if no shape is present on any of these layers.

`'cutClassSizes ({f_width | (f_width f_length) | t_name}`

(Virtuoso Layout Suite EXL and higher tiers) Specifies a list of `cutClass` sizes to be used for checking `'viaLayers` and `'exceptViaLayers`.

`'mask1 | 'mask2 | 'mask3`

(Virtuoso Layout Suite EXL and higher tiers) Specifies the color mask of the shapes for which the rule is applicable.

`'endToEndGapRanges g_endToEndGapRanges`

(Virtuoso Advanced Node Option for GAA and Virtuoso Layout Suite EXL and higher tiers) The constraint applies when either side end-to-end gap with another wire is within these ranges.

`'otherEndToEndGapRanges g_otherEndToEndGapRanges`

(Virtuoso Advanced Node Option for GAA and Virtuoso Layout Suite EXL and higher tiers) The constraint applies when one side end-to-end gap with another wire is within the `endToEndGapRanges` ranges and the other side end-to-end gap with another wire is within the `otherEndToEndGapRanges` ranges.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`f_default`

The length value to be used when no table entry applies.

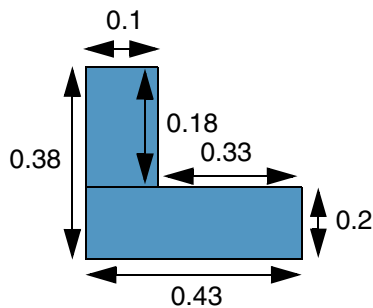
Example

The allowed lengths for rectangles on Metal1 are determined as follows:

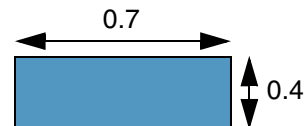
- If width is 0.01, the length must be greater than or equal to 0.1 and less than or equal to 0.18.
- If width is 0.02, the length must be greater than or equal to 0.3 and less than or equal to 0.43.
- If width is 0.04, the length must be greater than or equal to 0.3.

```
spacingTables (
  ( allowedLengthRanges "Metal1"
    (( "width" nil nil ))
    (
      0.01 ("[0.1 0.18]")
      0.02 ("[0.3 0.43]")
      0.04 (">=0.3")
    )
  )
) ;spacingTables
```

■ Metal1



a) PASS. The length of the rectangle with width 0.01 is 0.18, which falls in the range [0.1 0.18], and the length of the rectangle with width 0.2 is 0.43, which falls in the range [0.3 0.43].



b) PASS. The length of the rectangle with width 0.04 is 0.07 (>0.3).

allowedNeighborWidthRanges (One layer)

```
spacingTables(
  ( allowedNeighborWidthRanges tx_layer
    (( "width" nil nil )
      'distanceWithin f_within
      ['widthRanges (g_widthRanges)]
      ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
          ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) Specifies the width of a shape that is within a certain distance of a neighboring shape on the same layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of the <i>table</i> row is as follows: (<i>f_width</i> (<i>g_ranges</i>)) where, ■ <i>f_width</i> is the width of the neighboring shape. ■ <i>g_ranges</i> is a list of the allowed width <u>ranges</u>, applied when the width of the neighboring shape is greater than or equal to the corresponding index. Type: A 1-D table specifying floating point width values

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

`'distanceWithin f_within`

The constraint applies only if the distance between the shapes is less than this value.

`'widthRanges g_widthRanges`

The constraint applies only if the widths of the shapes on the specified layer falls in these width ranges.

Type: Floating-point values specifying width ranges

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see Region-based Rule (One layer).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`f_default`

The width value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Example

Two neighboring Metal1 shapes that are less than 1.5 apart and have widths less than or equal to 1.0 must satisfy the following conditions:

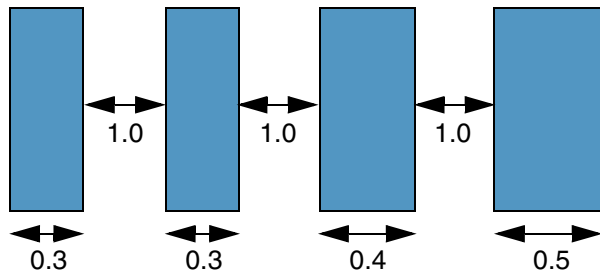
- If the width of a shape is greater than or equal to 0.3, the width of the neighboring shape must be 0.3 or 0.4.
- If the width of a shape is greater than or equal to 0.4, the width of the neighboring shape must be 0.3, 0.4, or 0.5.
- If the width of a shape is greater than or equal to 0.5, the width of the neighboring shape must be 0.4 or 0.5.

Virtuoso Technology Data Constraints Reference

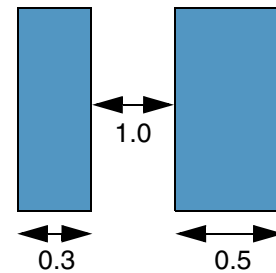
Length and Width Constraints

```
spacingTables (
  ( allowedNeighborWidthRanges "Metal1"
    (( "width nil nil )
      'distanceWithin 1.5
      'widthRanges ("[0 1.0]")
    )
    (
      0.3 (0.3 0.4)
      0.4 (0.3 0.4 0.5)
      0.5 (0.4 0.5)
    )
  )
) ;spacingTables
```

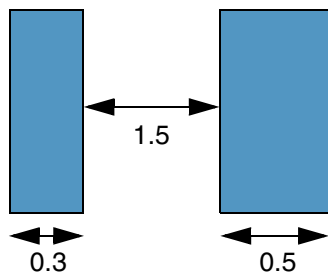
■ Metal1



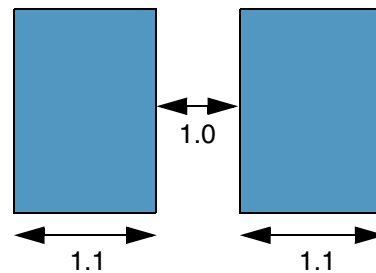
a) PASS. The width of all shapes is in the range [0 1.0] and the distance between each pair of shapes is 1.0 (<1.5). Each shape has neighboring shapes that satisfy the width requirement.



b) FAIL. The shape with width 0.3 can have a neighboring shape with width 0.3 or 0.4 and a shape with width 0.5 can have a neighboring shape with width 0.4 or 0.5.



c) The constraint does not apply because the distance between the shapes is 1.5 (and not less than 1.5).



d) The constraint does not apply because the width of both shapes is 1.1, which lies outside the range [0 1.0].

allowedNeighborWidthRanges (Two layers)

```
spacingTables (
  ( allowedNeighborWidthRanges tx_layer1 tx_layer2
    (( "width" nil nil )
      'distanceWithin f_within
      ['widthRanges (g_widthRanges)]
      ['otherWidthRanges (g_otherWidthRanges)]
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) Specifies the width of a *layer1* shape that is within a certain distance of a neighboring *layer2* shape.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of the <i>table</i> row is as follows: (<i>f_width</i> (<i>g_ranges</i>)) where, ■ <i>f_width</i> is the width of the neighboring shape on <i>layer2</i>. ■ <i>g_ranges</i> is a list of the allowed width <u>ranges</u> for the shape on <i>layer1</i>, applied when the width of the neighboring <i>layer2</i> shape is greater than or equal to the corresponding index. Type: A 1-D table specifying floating point width values

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

`'distanceWithin f_within`

The constraint applies only if the distance between the shapes is less than this value.

`'widthRanges g_widthRanges`

The constraint applies only if the width of the *layer1* shape falls in these width ranges.

Type: Floating-point values specifying width ranges

`'otherWidthRanges (g_otherWidthRanges)`

The constraint applies only if the width of the *layer2* shape falls in these width ranges.

Type: Floating-point values specifying width ranges

`f_default`

The width value to be used when no table entry applies.

Example

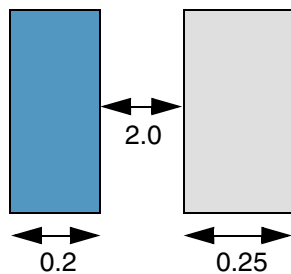
If a Metal2 shape with width greater than or equal to 0.25 is at a distance less than 2.5 from a Metal1 shape, then the width of the Metal1 shape must be greater than or equal to 0.2 and less than or equal to 0.3.

Virtuoso Technology Data Constraints Reference

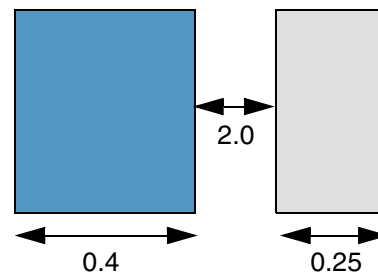
Length and Width Constraints

```
spacingTables (
  ( allowedNeighborWidthRanges "Metal1" "Metal1"
    (( "width nil nil )
      'distanceWithin 2.5
    )
    (
      0.25 ("[0.2 0.3]")
    )
  )
) ;spacingTables
```

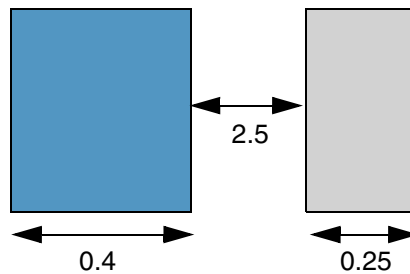
Metal2
 Metal1



a) PASS. The distance between the two shapes is 2.0 (< 2.5). The width of the Metal2 shape is 0.25 and the width of the Metal1 shape is in the range [0.2 0.3].



b) FAIL. The distance between the two shapes is 2.0 (< 2.5) and the width of the Metal2 shape is 0.25. However, the width of the Metal1 shape is 0.4, which lies outside the range [0.2 0.3].



c) The constraint does not apply because the distance between the shapes is 2.5 (and not less than 2.5).

allowedNeighborWidthRangesOver

```
spacingTables (
  ( allowedNeighborWidthRangesOver tx_layer1 tx_layer2 tx_layer3
    (( "width" nil nil )
      ['distanceWithin f_within]
      ['widthRanges g_widthRanges]
      ['otherWidthRanges g_otherWidthRanges]
      [g_default]
    )
    (g_table)
  )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) Specifies the width of a *layer1* shape that is optionally within a certain distance of a neighboring *layer2* shape, when both *layer1* and *layer2* shapes overlap a shape on *layer3*.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer3</i>	The third layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

g_table The format of the *table* row is as follows:

(*f_width* (*g_ranges*))

where,

- *f_width* is the width of the neighboring shape on *layer2*.
- *g_ranges* is a list of the allowed width ranges for the shape on *layer1*, applied when the width of the neighboring *layer2* shape is greater than or equal to the corresponding index.

Type: A 1-D table specifying floating point width values

Parameters

'distanceWithin *f_within*

The constraint applies only if the distance between the shapes is less than this value.

'widthRanges *g_widthRanges*

The constraint applies only if the width of the *layer1* shape falls in these width ranges.

Type: Floating-point values specifying width ranges

'otherWidthRanges (*g_otherWidthRanges*)

The constraint applies only if the width of the *layer2* shape falls in these width ranges.

Type: Floating-point values specifying width ranges

f_default The width value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

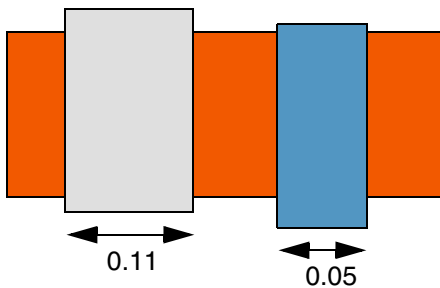
Length and Width Constraints

Example

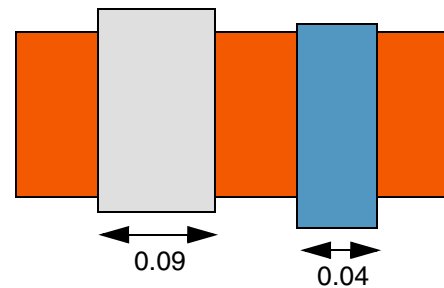
The width of a Metal1 shape must be 0.04, 0.05, or 0.06. If neighboring Metal1 and Metal2 shapes overlap a Metal3 shape and the width of a Metal2 shape is greater than or equal to 0.1, the width of the Metal1 shape must be 0.05 or 0.06.

```
spacings (
  ( allowedWidthRanges "Metal1" (0.04 0.05 0.06))
) ;spacings
spacingTables (
  ( allowedNeighborWidthRangesOver "Metal1" "Metal2" "Metal3"
    (( "width" nil nil )
      'otherWidthRanges (">=0.1")
    )
    ( 0.1 (0.05 0.06))
  )
) ;spacingTables
```

■ Metal3
■ Metal2
■ Metal1



a) PASS. The width of the Metal2 shape is 0.11 (>0.1) and the Metal1 shape has the allowed width.



b) PASS. The `allowedNeighborWidthRangesOver` constraint does not apply because the width of the Metal2 shape is 0.09 (<0.1). The width of the Metal1 shape is 0.04, which satisfies the `allowedWidthRanges` constraint.

allowedPRBoundaryDimensions

```
spacings (
    ( allowedPRBoundaryDimensions
      ['horizontal | 'vertical]
      ['stepSize f_stepSize]
      (g_ranges)
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the valid dimensions for a PR boundary object. A PR boundary can be a rectilinear object. The dimension is defined as the distance between a pair of facing edges, in the specified direction.

Values

g_ranges The allowed PR boundary dimension ranges.
Type: Floating-point values specifying dimension ranges

Parameters

'horizontal | 'vertical The PR boundary dimension is measured in this direction.
Type: Boolean

'stepSize f_stepSize The allowed dimensions (D_{legal}) are restricted to the following values, up to the upper bound of each range:

$$D_{\text{legal}} = D_{\text{lb}} + n * \text{stepSize}$$

where, D_{lb} is the lower bound of each range and $n \geq 0$.

If the lower bound is not included in the range specification, the computation is performed using the lower bound, but the first value calculated, which is equal to the lower bound, is not considered a legal value. For a rectilinear PR boundary, all dimensions in the specified direction must satisfy the constraint. If multiple ranges are specified, 'stepSize applies to all ranges.

Virtuoso Technology Data Constraints Reference

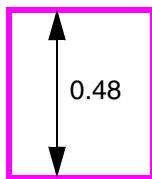
Length and Width Constraints

Example

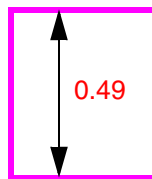
The vertical distance between the facing edges of a PR boundary must be greater than or equal to 0.24 and less than or equal to 4.8. A legal dimension value is the sum of the lower bound of this distance range (0.24) and a multiple of 0.048.

```
spacings(  
  ( allowedPRBoundaryDimensions  
    'vertical  
    'stepSize 0.048  
    ("[0.24 4.8]")  
  )  
) ;spacings
```

□ PR boundary



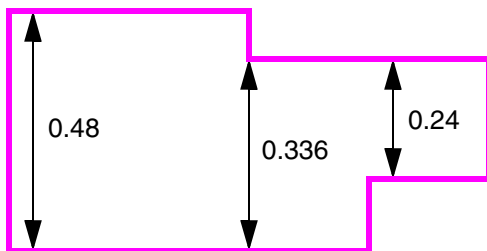
a) PASS. The vertical 0.48 distance (for n=5) is an allowed value that falls in the specified dimensions range.



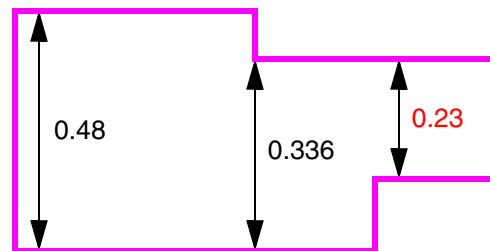
b) FAIL. The distance 0.49 is not a legal value.



c) FAIL. The distance of 0.192 is less than the 0.24, the lower bound.



d) PASS. All vertical distances are legal values (for n = 0, 2, and 5) and fall in the specified dimensions range.



e) FAIL. The distance 0.23 is less than 0.24, the lower bound.

allowedSpanLengthRanges

```
spacings(  
    ( allowedSpanLengthRanges tx_layer  
      ['any | 'horizontal | 'vertical]  
      ['mask1 | 'mask2 | 'mask3]  
      ['orthogonalLength f_length]  
      ['otherSpanLength f_spanLength  
        | 'otherSpanTable g_1DSpanLengthTable]  
      (g_ranges)  
    )  
); spacings
```

Specifies all allowable span lengths on a routing layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The span lengths that are allowed. Type: Floating-point values specifying a <u>range</u> of span lengths

Parameters

'any 'horizontal 'vertical	The direction in which the span length is measured. Type: Boolean
'mask1 'mask2 'mask3	The mask that the triggering wires of this rule belong to. Type: Boolean
'orthogonalLength <i>f_length</i>	The distance between two inside-facing corners must be greater than or equal to this value.
'otherSpanLength <i>f_spanLength</i> 'otherSpanTable <i>g_1DSpanLengthTable</i>	

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

(Virtuoso Advanced Node for Layout Only) The constraint applies only when the `rectShapeDir` constraint is specified for the layer and if either of the following conditions applies:

- The span length measured in a direction perpendicular to the specified direction is greater than or equal to *f_spanLength*.
- The span length measured in a direction orthogonal to the specified direction is less than or equal to the table with the *g_1DSpanLengthTable* index. The table is valid only for indexes below the last row. The parameter does not apply for orthogonal span lengths greater than or equal to the index of the last table row.

Examples

- [Example 1: allowedSpanLengthRanges with horizontal, vertical, and orthogonalLength](#)
- [Example 2: allowedSpanLengthRanges with horizontal, vertical, and otherSpanLength](#)
- [Example 3: allowedSpanLengthRanges with horizontal, vertical, and otherSpanTable](#)

Example 1: allowedSpanLengthRanges with horizontal, vertical, and orthogonalLength

The constraint is satisfied if the following conditions are met for a rectilinear object on Metal2:

- The span length measured vertically is either equal to 0.1, 0.15, or 0.2 or greater than 0.3 and less than 15, and the distance between two inside-facing corners is greater than or equal to 0.13.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

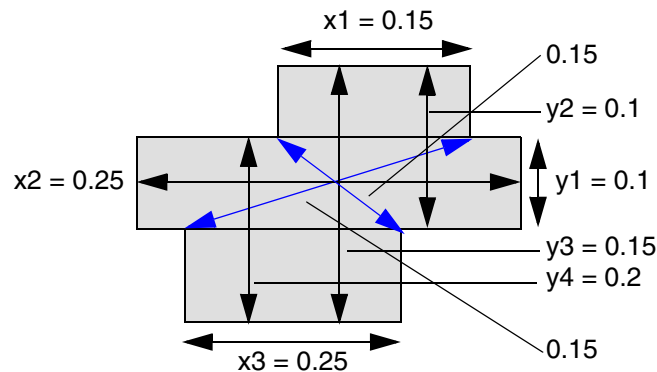
- The span length measured horizontally is either equal to 0.15 or 0.25 or greater than 0.4 and less than 15.

```

spacings(
  ( allowedSpanLengthRanges "Metal2"
    'vertical
    'orthogonalLength 0.13
    (0.1 0.15 0.2 "(0.3 15) ")
  )
  ( allowedSpanLengthRanges "Metal2"
    'horizontal
    (0.15 0.25 "(0.4 15) ")
  )
) ;spacings

```

□ Metal2



PASS. The vertical span lengths y_1 , y_2 , y_3 , and y_4 are 0.1, 0.1, 0.15, 0.2, respectively, and the distance between both pairs of inside-facing corners is 0.15 (>0.13), as indicated by blue arrows. Additionally, the horizontal span lengths for x_1 , x_2 , and x_3 are 0.15, 0.25, and 0.25, respectively.

Example 2: *allowedSpanLengthRanges* with horizontal, vertical, and otherSpanLength

The constraint is satisfied if the following conditions are met for a rectangular object on Metal2:

- The span length measured vertically is either equal to 0.05, 0.06, 0.07, 0.08 or 0.09 or greater than 0.1 and less than 15.

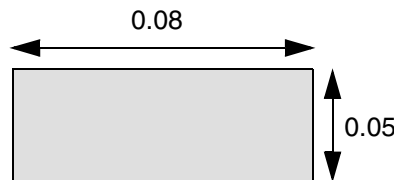
Virtuoso Technology Data Constraints Reference

Length and Width Constraints

- The span length measured horizontally is either equal to 0.07 or greater than 0.1 and less than 15. However, this condition is checked only if the span length measured vertically is greater than 0.05.

```
spacings(  
  ( allowedSpanLengthRanges "Metal2"  
    'vertical  
    (0.05 0.06 0.07 0.08 0.09 "(0.1 15)")  
  )  
  ( allowedSpanLengthRanges "Metal2"  
    'horizontal  
    'otherSpanLength 0.05  
    (0.07 "(0.1 15)")  
  )  
) ;spacings
```

□ Metal2



PASS. The first constraint is met because the vertical span length is 0.05. The second constraint does not apply because the vertical span length is equal to 0.05 (and not greater than 0.05). A violation would occur if 'otherSpanLength was not specified.

Example 3: allowedSpanLengthRanges with horizontal, vertical, and otherSpanTable

The constraint is satisfied if the following conditions are met for a rectangular object on Metal2:

- The span length measured vertically is either equal to 0.04, 0.05, 0.06, 0.07, 0.08, or 0.09 or greater than 0.1 and less than 15.
- The span length measured horizontally is either equal to 0.07 or greater than 0.1 and less than 15, except in these cases:
 - If the vertical edge is less than or equal to 0.04, the horizontal span length must be greater than or equal to 0.09.

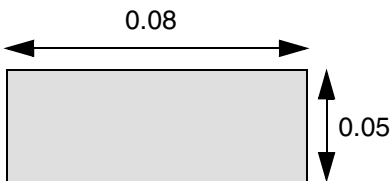
Virtuoso Technology Data Constraints Reference

Length and Width Constraints

- ❑ If the vertical edge is less than or equal to 0.06, the horizontal span length must be greater than or equal to 0.08.

```
spacings(
    (allowedSpanLengthRanges "Metal2"
      'vertical 'mask1
      (0.04 0.05 0.06 0.07 0.08 0.09 "(0.1 15)"))
    (allowedSpanLengthRanges "Metal2"
      'horizontal 'mask1
      'otherSpanTable ((( "spanlength"))
        (
          0.00      0.09
          0.0405    0.08
          0.0605    0.00
        ) )
      (0.07 "(0.1 15)"))
  ) ;spacings
```

☐ Metal2



PASS. The vertical span length of 0.05 fulfills the first span length table. The horizontal span length of 0.08 is checked against the otherSpanTable entry corresponding to the vertical span length of 0.05. It passes the corresponding requirement of greater than or equal to 0.08.

allowedWidthRanges

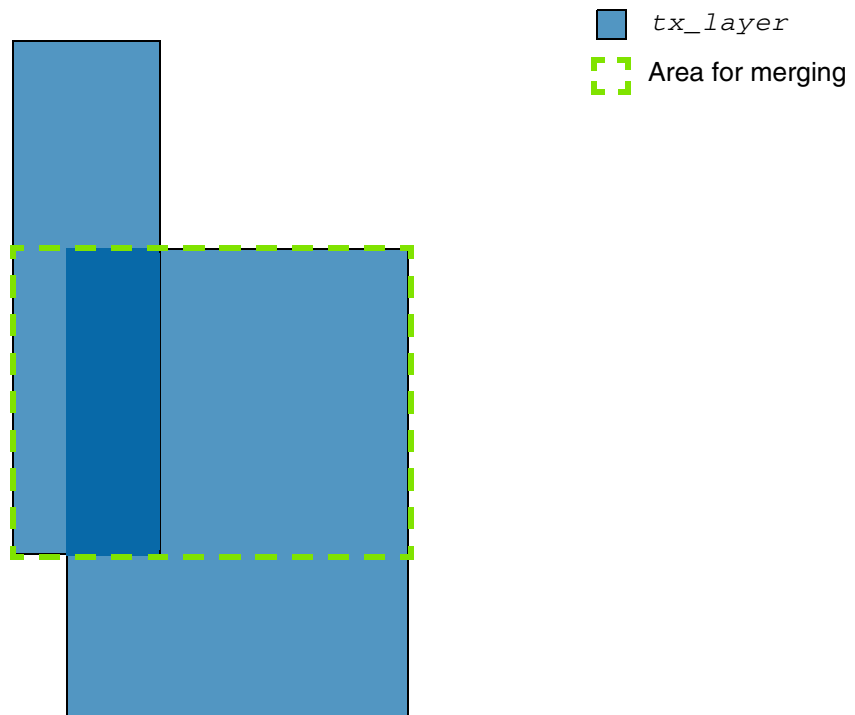
```

spacings (
  ( allowedWidthRanges tx_layer
    ['length f_length]
    ['orthogonal]
    ['horizontal | 'vertical | 'measureHorizontal | 'measureVertical]
    ['stepSize f_stepSize ['stepRange g_range]]
    ['exceptOverlap f_exceptOverlap]
    ['mask1 | 'mask2 | 'mask3]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    (g_ranges)
  )
) ;spacings

```

Specifies the allowed width values for shapes on a layer. Polygon shapes are broken down into rectangles and the shorter dimension of each rectangle is checked against the allowed width values.

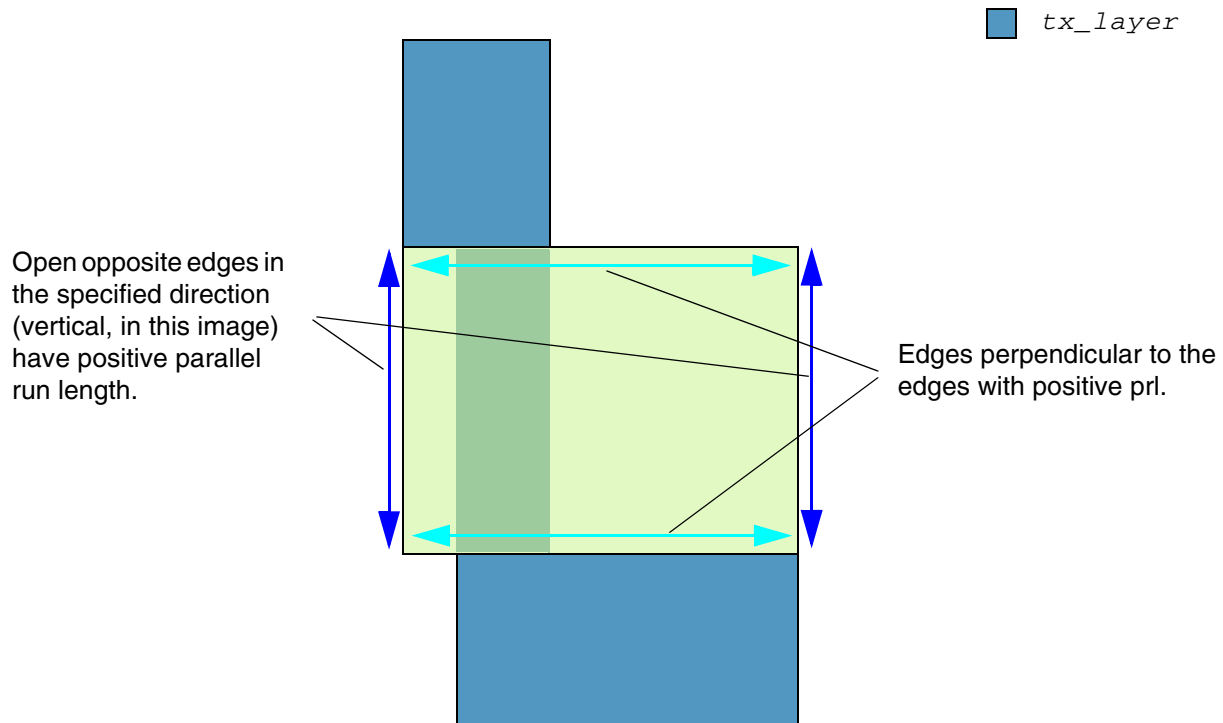
Consider two partially overlapping shapes and the area considered for merging.



Virtuoso Technology Data Constraints Reference

Length and Width Constraints

The constraint checks if the open opposite edges in the specified direction have positive parallel run length and the constraint is applied perpendicular to these edges.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The allowed width ranges. Type: Floating-point values specifying width <u>ranges</u>

Parameters

<code>'length f_length</code>	The constraint applies only if the length of the shape is greater than or equal this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

`'orthogonal`

The distance between two inside corners of a rectilinear shape measured in the direction orthogonal to the direction specified for the constraint must be greater than or equal to the smallest specified width value.

Type: Boolean

`'horizontal | 'vertical | 'measureHorizontal | 'measureVertical`

The parameters `'horizontal` and `'vertical`, if specified, indicate the direction in which width is measured.

(Virtuoso Advanced Node for Layout Standard) If `'measureHorizontal` or `'measureVertical` is specified, the width is always measured in this direction, regardless of whether this direction is the width or length of the shape.

Note: Do not mix direction types within a constraint group. Specify either `'horizontal/'vertical` or `'measureHorizontal/'measureVertical`.

Type: Boolean

`'stepSize f_stepSize`

(Virtuoso Advanced Node for Layout Standard) The allowed widths (W_{legal}) are restricted to the following values:

$$W_{\text{legal}} = W_{\text{lb}} + n * \text{stepSize}$$

where, W_{lb} is the lower bound of the range specified by `stepRange` and $n \geq 0$. If `stepRange` is not specified, W_{lb} is the lower bound of the constraint value.

`'stepRange g_stepRange`

(Virtuoso Advanced Node for Layout Standard) The stepped width specified with `'stepSize` applies only in this range, which is fully contained in the constraint value.

Type: Floating-point values specifying a width range

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

`'exceptOverlap f_exceptOverlap`

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply to merged areas whose overlap length is less than or equal to this value.

The overlap length of the merged area is measured in the direction of the wires, which is perpendicular to the width of the wires. Both merging wires must be in the same direction.

`'mask1 | 'mask2 | 'mask3`

(Virtuoso Layout Suite EXL and higher tiers) The mask of the shapes to which this rule belongs to.

Type: Boolean

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Examples

- Example 1: allowedWidthRanges
- Example 2: allowedWidthRanges with vertical and orthogonal
- Example 3: allowedWidthRanges with measureVertical and stepSize
- Example 4: allowedWidthRanges with stepSize and stepRange
- Example 5: allowedWidthRanges with measureVertical and measureHorizontal
- Example 6: allowedWidthRanges with vertical and horizontal
- Example 7: allowedWidthRanges with exceptOverlap

Virtuoso Technology Data Constraints Reference

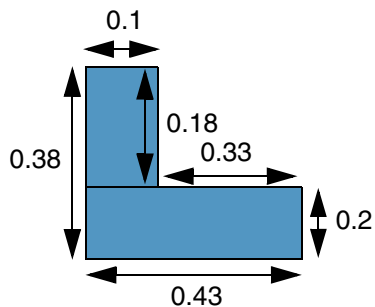
Length and Width Constraints

Example 1: *allowedWidthRanges*

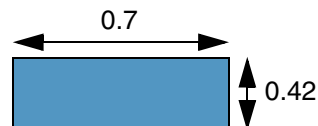
The allowed widths for shapes on Metal1 are 0.1, 0.15, 0.2, 0.25, and 0.3. Additionally, the width of a shape can be greater than 0.4 and less than 1.2.

```
spacings (  
  ( allowedWidthRanges "Metal1"  
    (0.1 0.15 0.2 0.25 0.3 "(0.4 1.2)")  
  )  
) ;spacings
```

■ Metal1



a) PASS. The widths of the two rectangles are 0.1 and 0.2.



b) PASS. The width of the rectangle is 0.42, which falls in the range (0.4 1.2).

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

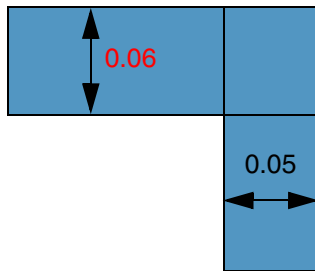
Example 2: *allowedWidthRanges* with vertical and orthogonal

The vertical width of a wire must be equal to 0.05, 0.07, 0.1, 0.12, or 0.15. If two wires overlap or touch, the horizontal distance between two inside corners of the wires must be greater than or equal to 0.05.

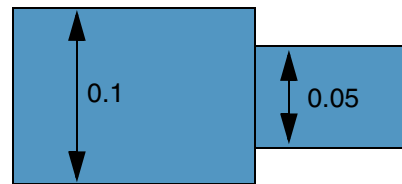
```
spacings (
  ( allowedWidthRanges "Metal1"
    'vertical
    'orthogonal
    (0.05 0.07 0.1 0.12 0.15)
  )
) ;spacings
```

■ Metal1

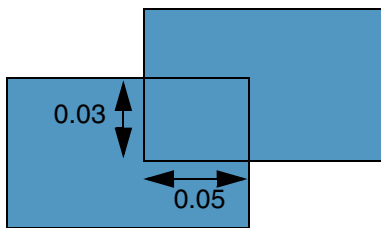
Preferred routing direction: Vertical



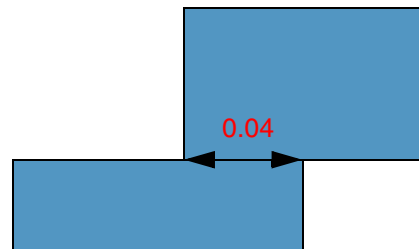
a) FAIL. The width 0.06 measured vertically is not an allowed width.



b) PASS. Both 0.05 and 0.1 are allowed vertical widths.



c) PASS. The horizontal width of 0.05 measured between the two inside corners is an allowed value. The constraint is satisfied even though the vertical width of 0.03 is not an allowed value.



d) FAIL. The horizontal width must be an allowed value because the wires touch. However, 0.04 is not an allowed value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Example 3: *allowedWidthRanges* with *measureVertical* and *stepSize*

The vertical width of a wire must be greater than or equal to 0.1 and less than or equal to 0.2. A legal width value is also the sum of the lower bound of this width range (0.1) and a multiple of 0.05.

```
spacings(  
  ( allowedWidthRanges "Metal1"  
    'measureVertical  
    'stepSize 0.05  
    ("[0.1 0.2]")  
  )  
) ;spacings
```

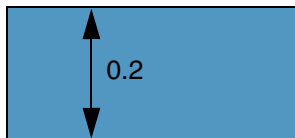
■ Metal1



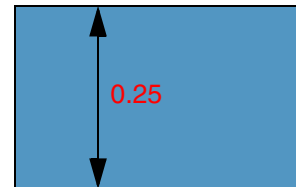
a) PASS. The constraint is satisfied for n=0.



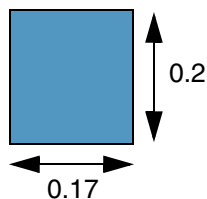
b) PASS. The constraint is satisfied for n=1.



c) PASS. The constraint is satisfied for n=2.



d) FAIL. The maximum allowed width is 0.2.



e) PASS. The constraint applies in the vertical direction and is satisfied for n=2.

Virtuoso Technology Data Constraints Reference

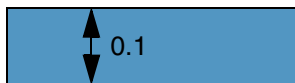
Length and Width Constraints

Example 4: *allowedWidthRanges* with *stepSize* and *stepRange*

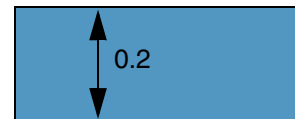
The vertical width of a wire must be greater than or equal to 0.1 and less than or equal to 0.4. A legal width value is also the sum of the lower bound of this width range (0.1) and a multiple of 0.05. For widths greater than or equal to 0.1 and less than or equal to 0.21, only discrete widths at a 0.05 step are allowed.

```
spacings(  
  ( allowedWidthRanges "Metal1"  
    'stepSize 0.05  
    'stepRange "[0.1 0.21]"  
    ("[0.1 0.4]")  
  )  
) ;spacings
```

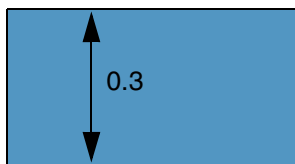
■ Metal1



a) PASS. The constraint is satisfied for n=0.



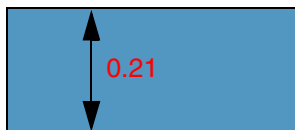
b) PASS. The constraint is satisfied for n=2.



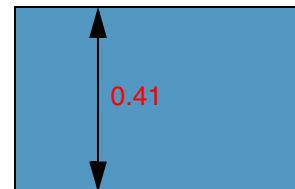
c) PASS. The constraint is satisfied for n=4.



d) FAIL. In the [0.1 0.21] range, only discrete widths at a 0.05 step are allowed.



e) FAIL. In the [0.1 0.21] range, only discrete widths at a 0.05 step are allowed.



f) FAIL. The maximum allowed width is 0.41.

Virtuoso Technology Data Constraints Reference

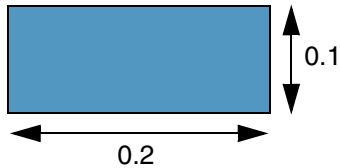
Length and Width Constraints

Example 5: *allowedWidthRanges* with *measureVertical* and *measureHorizontal*

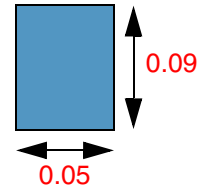
Two `allowedWidthRanges` constraints are defined in an AND constraint group. A check is performed in the specified direction, regardless of whether this direction is the width or length of the shape.

```
spacings (
  ( allowedWidthRanges "Metal1"
    'measureVertical
    (">=0.1")
  )
  ( allowedWidthRanges "Metal1"
    'measureHorizontal
    (">=0.2")
  )
) ;spacings
```

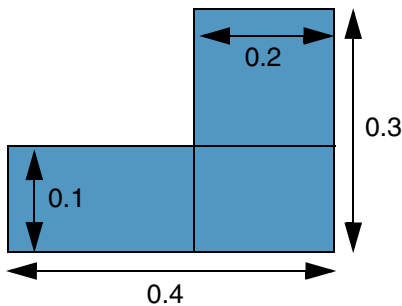
■ Metal1



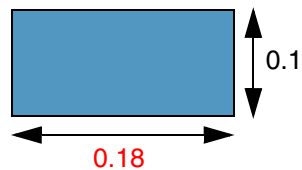
a) PASS. The constraint is satisfied in both directions.



b) FAIL. The constraint is violated in both directions.



c) PASS. The constraint is satisfied in both directions for each rectangle.



d) FAIL. The constraint is violated in the horizontal direction.

Virtuoso Technology Data Constraints Reference

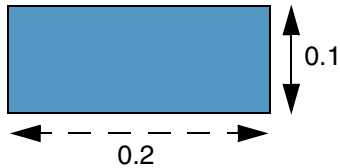
Length and Width Constraints

Example 6: *allowedWidthRanges* with vertical and horizontal

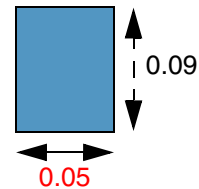
Two `allowedWidthRanges` constraints are defined in an AND constraint group. A check is performed in the specified direction for the shape width, which is the shorter of the two dimensions. A check is not performed on the length represented by the dashed arrows.

```
spacings (
  ( allowedWidthRanges "Metal1"
    'vertical
    (">=0.1")
  )
  ( allowedWidthRanges "Metal1"
    'horizontal
    (">=0.2")
  )
) ;spacings
```

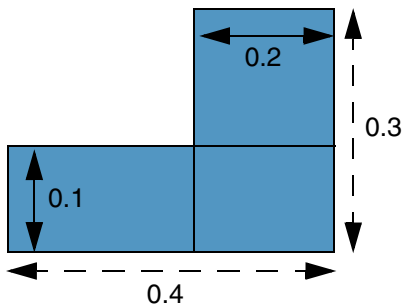
■ Metal1



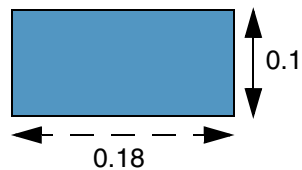
a) PASS. The constraint is satisfied in the vertical direction.



b) FAIL. The constraint is violated in the horizontal direction.



c) PASS. The constraint is satisfied in both directions for each rectangle.



d) PASS. The constraint is satisfied in the vertical direction.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

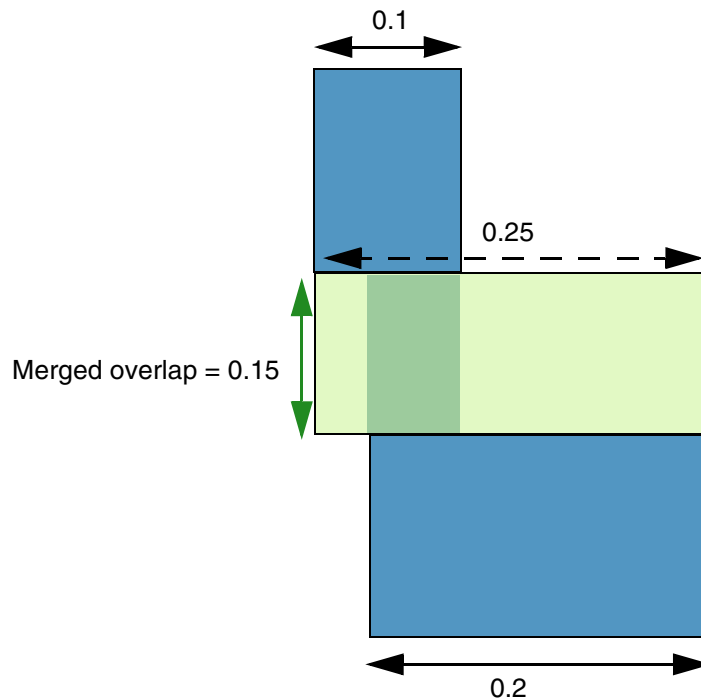
Example 7: *allowedWidthRanges* with *exceptOverlap*

Consider the following cases where two `allowedWidthRanges` constraints are defined in an AND constraint group. A check is performed in the specified direction for the shape width that is the shorter of the two dimensions. A check is not performed on the length represented by the dashed arrows. The constraints do not apply to merged areas whose overlap length is less than or equal to the `exceptOverlap` value.

In the following case, the constraint passes with or without the `exceptOverlap` parameter. The merged overlap value is 0.15, which is equal to the `exceptOverlap` value, 0.15.

```
spacings(  
  ( allowedWidthRanges "Metal1"  
    'horizontal  
    'exceptOverlap 0.15  
    (0.1 0.2 0.3)  
  )  
  ( allowedWidthRanges "Metal1"  
    'vertical  
    'exceptOverlap 0.05  
    (0.4 0.5)  
  )  
) ;spacings
```

■ Metal1
■ Merged area



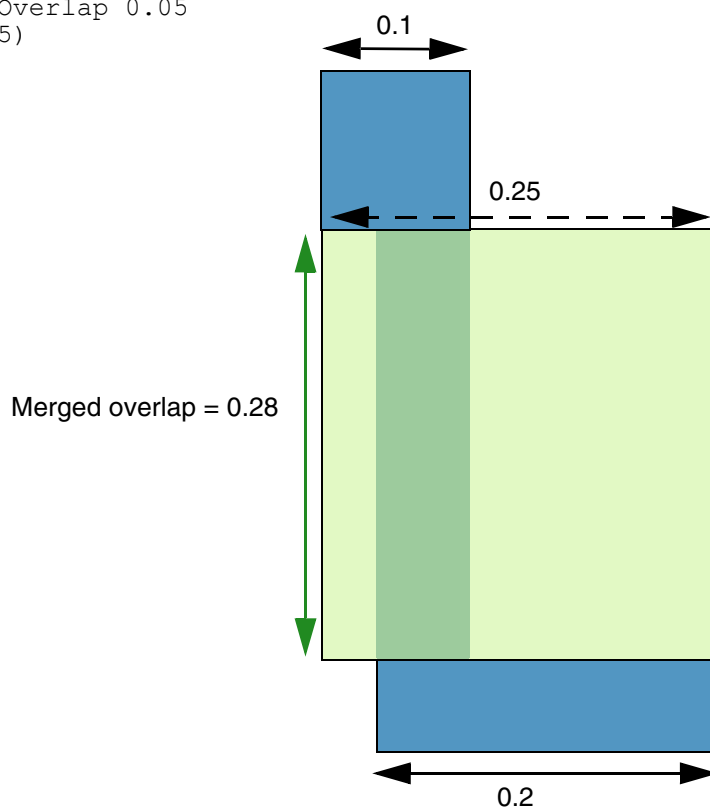
Virtuoso Technology Data Constraints Reference

Length and Width Constraints

In the following case, the constraint passes. The merged overlap value is 0.28, which is more than the exceptOverlap value, 0.27.

```
spacings(  
  ( allowedWidthRanges "Metal1"  
    'horizontal  
    'exceptOverlap 0.27  
    (0.1 0.2 0.25 0.3)  
  )  
  ( allowedWidthRanges "Metal1"  
    'vertical  
    'exceptOverlap 0.05  
    (0.4 0.5)  
  )  
) ;spacings
```

■ Metal1
■ Merged area



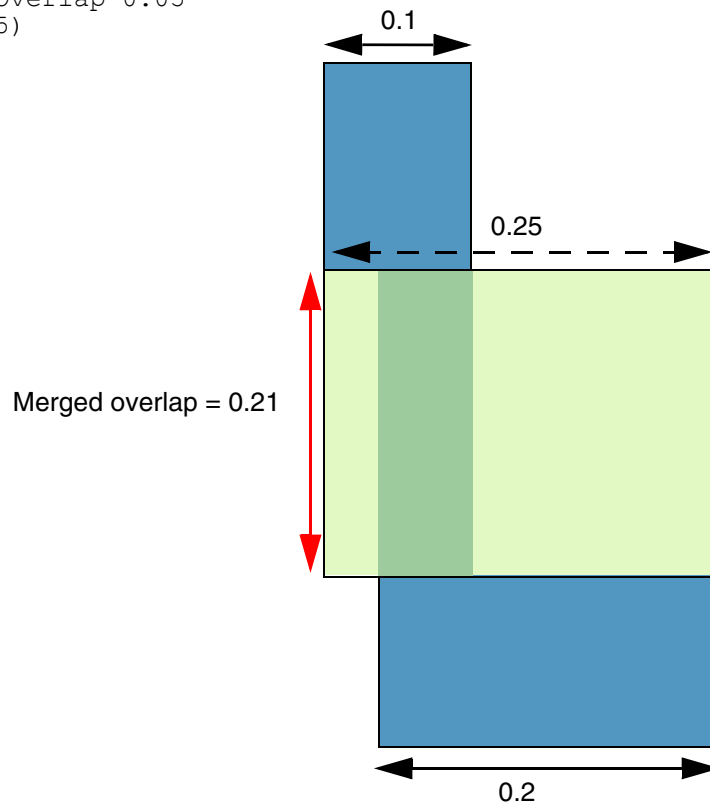
Virtuoso Technology Data Constraints Reference

Length and Width Constraints

In the following case, the constraint fails. The merged overlap value is 0.21, which is more than the exceptOverlap value, 0.19.

```
spacings(  
  ( allowedWidthRanges "Metal1"  
    'horizontal  
    'exceptOverlap 0.19  
    (0.1 0.2 0.23 0.3)  
  )  
  ( allowedWidthRanges "Metal1"  
    'vertical  
    'exceptOverlap 0.05  
    (0.4 0.5)  
  )  
) ;spacings
```

■ Metal1
■ Merged area



boundaryEOLBlockage

```

spacings (
    ( boundaryEOLBlockage tx_Layer
        'offset g_offset
        {'horizontal | 'vertical | 'any}
        ['prl g_parallelRunLength 'within g_within 'distance g_distance]
        g_size
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Specifies the size and direction of end-of-line blockages from design boundary edges.

Wires starting in the design interior and ending with an end-of-line edge in the blockage violate the constraint. On the other hand, wires with no end-of-line edge in the blockage or those that start in the periphery but end in the blockage do not violate the constraint.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_size</i>	The size of the blockages.

Parameters

<i>'offset g_offset</i>	End-of-line blockages start from the specified offset from the design boundary.
<i>'horizontal 'vertical 'any</i>	End-of-line blockages are formed along design boundary edges in the specified direction.
<i>'prl g_parallelRunLength 'within g_within 'distance g_distance</i>	(Virtuoso Layout Suite EXL and higher tiers) A search window is formed by extending the given <i>g_parallelRunLength</i> value at ends-of-lines on both sides and the <i>g_within</i> value in the orthogonal direction, unless neighbor wires on both sides of an end-of-line are less than the given <i>g_distance</i> value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Examples

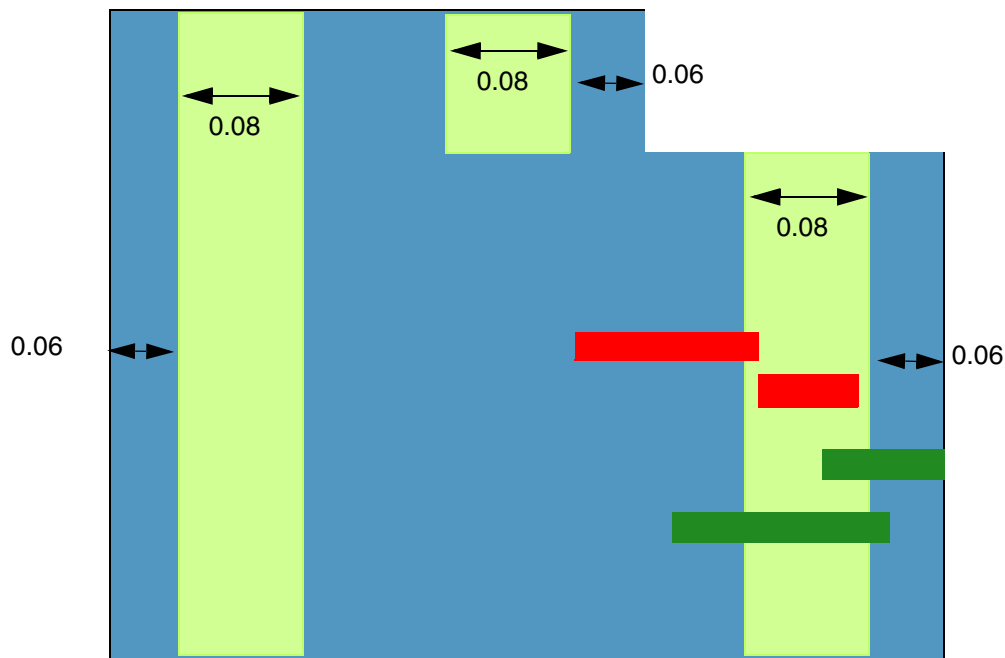
- Example 1: boundaryEOLBlockage with direction
- Example2: boundaryEOLBlockage with prl

Example 1: boundaryEOLBlockage with direction

For the Metal1 layer, the size of end-of-line blockages formed in the horizontal direction from the design boundary edges is 0.08.

```
spacings (  
    (boundaryEOLBlockage "Metal1"  
        'offset 0.06 'vertical  
        0.08)  
); spacings
```

■ Metal1
■ Blockages



a) FAIL. The red wire crossing the left edge of the blockage on the right boundary is a violation. The second red wire that is completely enclosed by the blockage is also a violation.

b) PASS. The first green wire crossing the right edge of the blockage and the second green wire going through the blockage are not violations.

Virtuoso Technology Data Constraints Reference

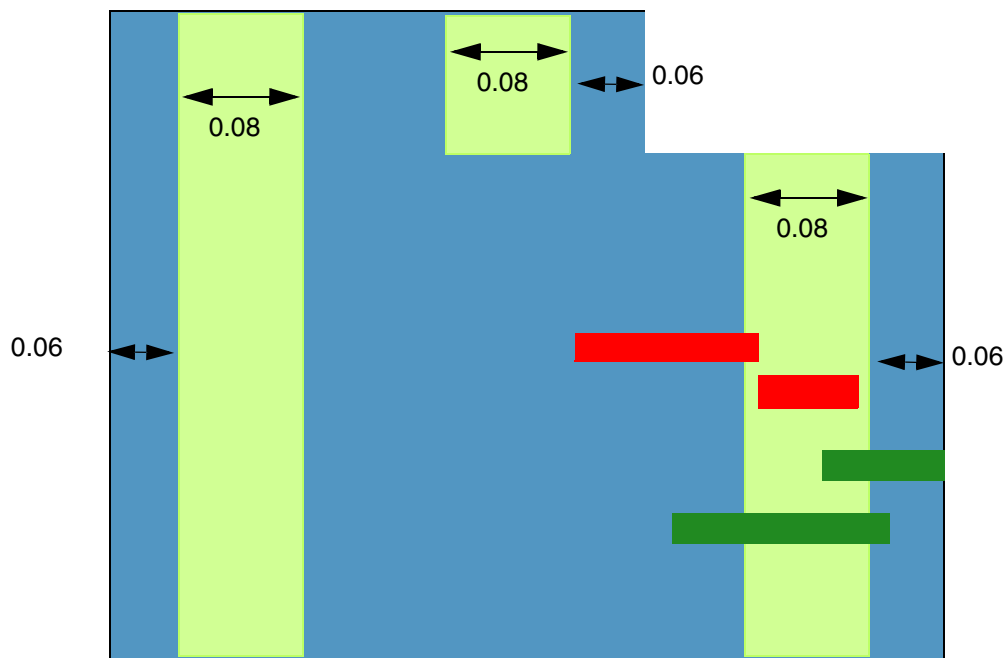
Length and Width Constraints

Example2: boundaryEOLBlockage with prl

For the Metal1 layer, the size of end-of-line blockages formed in the horizontal direction from the design boundary edges is 0.08. The search window is formed by extending the parallel run length by 0.05 on both sides and by 0.03 in the orthogonal direction, unless neighbor wires on both sides of the ends-of-lines are within 0.04.

```
spacings (
  (boundaryEOLBlockage "Metal1"
    'offset 0.06
    'vertical
    'prl 0.05 'within 0.03 'distance 0.04
    0.08)
) ;spacings
```

■ Metal1
■ Blockages



a) FAIL. The red wire crossing the left edge of the blockage on the right boundary is a violation. The second red wire that is completely enclosed by the blockage is also a violation.

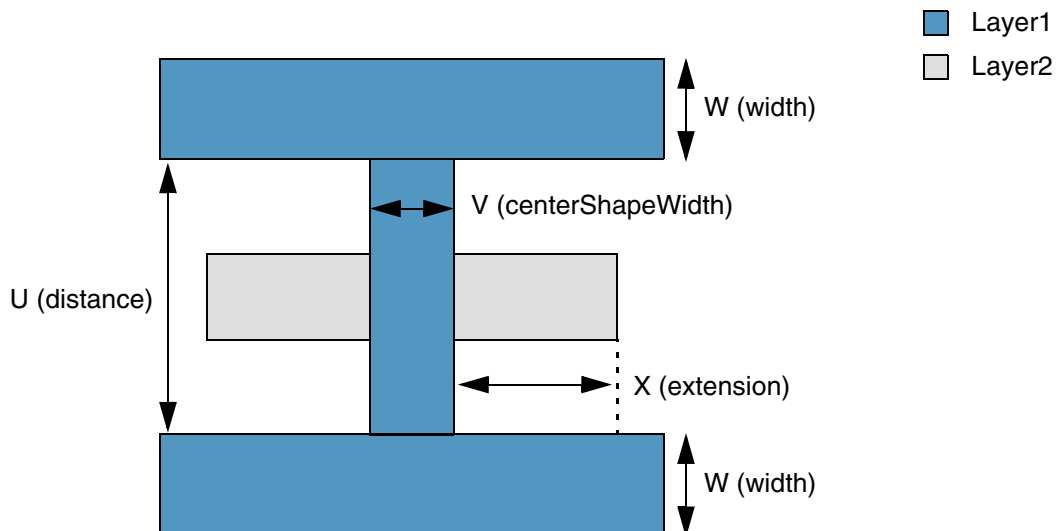
b) PASS. The first green wire crossing the right edge of the blockage and the second green wire going through the blockage are not violations.

illegalHGatePattern

```
orderedSpacings (  
  ( illegalHGatePattern tx_layer1 tx_layer2  
    'width f_width  
    'length f_length  
    'within f_distance  
    f_centershapeWidth  
  )  
) ;orderedSpacings
```

Specifies that shapes connected to form an H-shaped pattern on the same layer (*layer1*) cause a violation if any of the following conditions is true:

- The width of the rails (W) is less than the specified width (*width*).
- The extension of the rails (X) from the connecting shape is greater than the specified extension (*length*).
- The distance between the rails (U) is less than the specified distance (*distance*).
- The width (V) of the center shape is less than the specified constraint value (*centerShapeWidth*).



Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_centerShapeWidth</i>	The width of the center shape must be greater than or equal to this value.

Parameters

<code>'width f_width</code>	The width of the rails must be greater than or equal to this value.
<code>'length f_length</code>	The extension of the rails from the connecting shape must be less than or equal to this value.
<code>'within f_distance</code>	The distance between the rails must be greater than or equal to this value.

Example

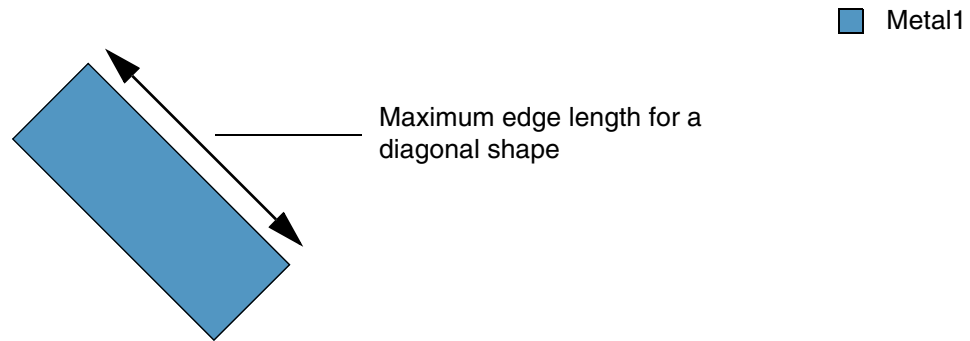
If shapes are connected to form an H-shaped pattern on Metal1, the width of the rails must be greater than or equal to 0.3, the extension of the rails from the connecting shape must be less than or equal to 0.9, the distance between the rails must be greater than or equal to 0.5, and the width of the center shape must be greater than or equal to 0.5.

```
orderedSpacings(  
  ( illegalHGatePattern "Metal1" "Metal2"  
    'width 0.3  
    'length 0.9  
    'within 0.5  
    0.5  
  )  
) ;orderedSpacings
```

maxDiagonalEdgeLength

```
spacings (
    ( maxDiagonalEdgeLength tx_layer f_length )
) ; spacings
```

Specifies the maximum edge length for a diagonal shape.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The maximum length of a diagonal edge.

Parameters

None

Example

The maximum diagonal edge length must be 3.0 on Metal1 and equal to the value of the technology parameter `diaglength2` on Metal2.

```
spacings (
    ( maxDiagonalEdgeLength "Metal1" 3.0 )
    ( maxDiagonalEdgeLength "Metal2" techParam("diaglength2") )
) ; spacings
```

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

maxDiffusionLength

```
orderedSpacings (
    ( maxDiffusionLength tx_layer1 tx_layer2
      ['width f_width]
      f_length
    )
) ;orderedSpacings
```

Specifies the maximum diffusion length between two cut shapes and between a cut shape and a line-end of the diffusion shape.

Values

<i>tx_layer1</i>	The first layer (diffusion) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer (cut) on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The distance between the edges of the shapes on <i>layer2</i> and between an edge of a <i>layer2</i> shape and the nearest <i>layer1</i> line-end must be less than this value.

Parameters

<i>'width f_width</i>	The constraint applies only if the width of the <i>layer1</i> shape is less than this value.

Virtuoso Technology Data Constraints Reference

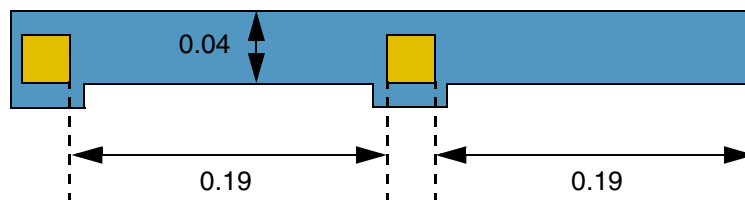
Length and Width Constraints

Example

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.

```
orderedSpacings(  
  ( maxDiffusionLength "Diffusion" "Contact"  
    'width 0.05  
    0.2  
  )  
) ;orderedSpacings
```

■ Contact
■ Diffusion



PASS. The width of the shape on the 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).

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

maxEdgeLength

```
spacings (
    ( maxEdgeLength tx_layer
        ['any | 'horizontalEdge | 'verticalEdge]
        ['allCorner | 'concaveCorner | 'convexCorner | 'mixedCorner]
        ['extendToAlign (g_width g_spacing)
            ['extendOnly | 'alignOnly]]
        ['width g_width]
        g_length
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the maximum edge length for a shape on a specified layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_length</i>	The length of each edge must be less than or equal to this value.

Parameters

'any 'horizontalEdge 'verticalEdge	The constraint applies only to the edges in the specified direction. The default is 'any. Type: Boolean
'allCorner 'concaveCorner 'convexCorner 'mixedCorner	The constraint applies to an edge between two concave corners, two convex corners, or a mixed (concave and convex) corner. The default is 'allCorner. Type: Boolean
'extendToAlign (g_width g_spacing) ['extendOnly 'alignOnly]	

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Specifies that if shapes on the upper and lower edges within a cell row have width equal to g_width , the shapes are extended and aligned. In addition, any gap on the same edge that is less than or equal to $g\_spacing$ is filled and merged while calculating the length.

Optionally, you can specify either of the following additional parameters:

- `extendOnly`: Only the extension operation is performed.
- `alignOnly`: Only the alignment operation is performed.

`'width  $g\_width$`

The constraint applies only on shapes with width equal to the given value.

Example

On layer `Implant1`, horizontal edges between two convex corners are at the most 0.08. Shapes with width 0.05 with a gap less than or equal to 0.07 are merged.

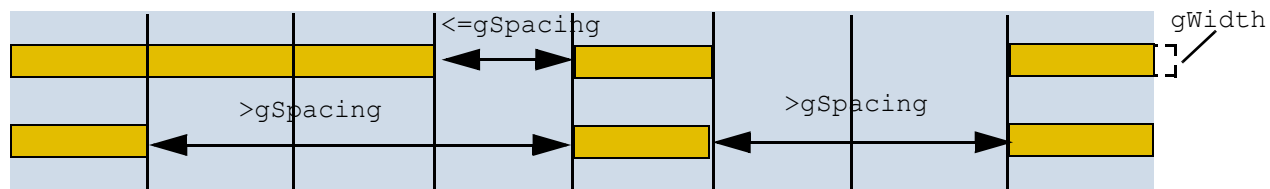
```
spacings (
  (maxEdgeLength "Implant1"
    'convexCorner
    'horizontalEdge
    'extendToAlign (0.05 0.07)
    0.08)
) ;spacings
```

Virtuoso Technology Data Constraints Reference

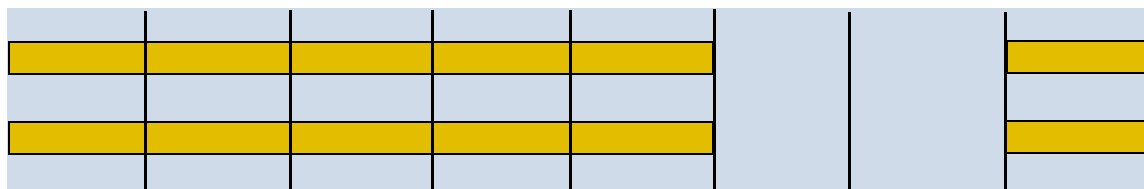
Length and Width Constraints

The following diagram illustrates `extendToAlign`.

Original shapes



After extension and alignment for length calculation



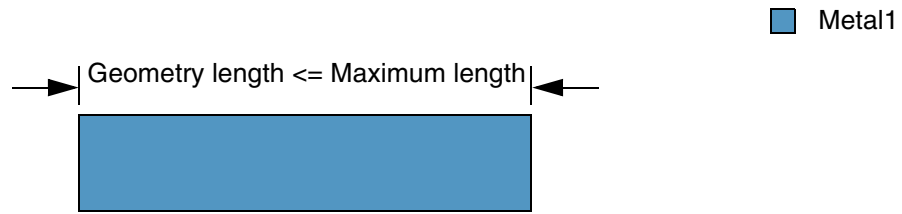
Virtuoso Technology Data Constraints Reference

Length and Width Constraints

maxLength

```
spacings (
    ( maxLength tx_layer ['width f_width] f_length )
) ;spacings
```

Sets the maximum orthogonal length for a shape on the specified layer.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The length of the shape must be less than or equal to this value.

Parameters

<i>'width f_width</i>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when the width of the shape is less than this value.

Example

The maximum length of a shape of width less than 0.08 on Metal1 must be 7.0.

```
spacings (
    ( maxLength "Metal1"
      'width 0.08
      7.0 )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

maxLengthWithCut

```
orderedSpacings (
  ( maxLengthWithCut tx_metal tx_cut
    'width g_width
    'cutWithin g_cutWithin
    'otherLayer g_otherLayer
    'otherWidth g_otherWidth
    'exceptOtherLength g_exceptOtherLength
    ['secondLayer g_secondLayer
      'secondWidth g_secondWidth
      'exceptSecondLength g_exceptSecondLength]
    g_length
  )
); orderedSpacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the maximum length required on a wire with a single cut without a neighbour cut within a specified distance.

Values

<i>tx_metal</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cut</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_length</i>	The length of each edge must be less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

'width *g_width* The constraint applies only if the width of the wire is less than this value.

'cutWithin *g_cutWithin*
The constraint applies if there is a neighbour cut within the given value.

'otherLayer *g_otherLayer*
The constraint specifies that if the wire length on the other layer is greater than this value, the rule is exempted.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

'otherWidth *f_otherWidth*

The constraint applies only if the wire width on the other layer is less than the this value.

'exceptOtherLength *g_exceptOtherLength*

This parameter specifies that if the wire length on the other layer is greater than this value, the rule is exempted.

'secondLayer *g_secondLayer*

The rule does not apply if there is a connected shape on this layer with length greater than *g_exceptSecondLength* and width less than *g_secondWidth*.

'secondWidth *g_secondWidth*

Used with *g_secondLayer*.

'exceptSecondLength *g_exceptSecondLength*

Used with *g_secondLayer*.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

maxNumMinEdges

```
spacings(  
    ( maxNumMinEdges tx_layer {x_numEdges | (f_length x_numEdges)} )  
); spacings
```

Sets the maximum number of consecutive small edges permitted in a step.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_numEdges</i>	The number of consecutive small edges in a step must be less than or equal to this value.

Parameters

<i>f_length</i>	The length of an edge in a set of consecutive small edges must be less than this value for it to be counted.

Virtuoso Technology Data Constraints Reference

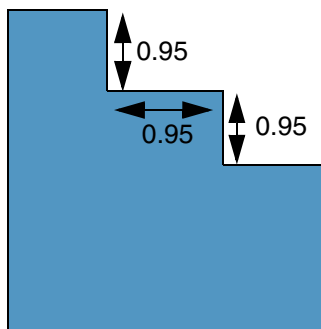
Length and Width Constraints

Example

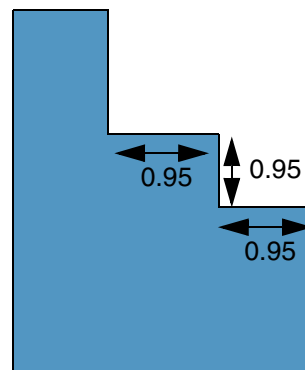
The maximum number of consecutive edges shorter than 1.0 must be 2.

```
spacings(  
  ( maxNumMinEdges "Metal1" (1.0 2) )  
) ; spacings
```

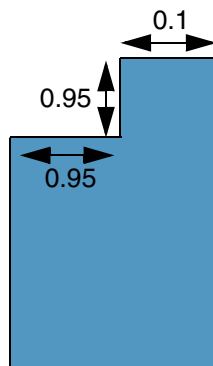
■ Metal1



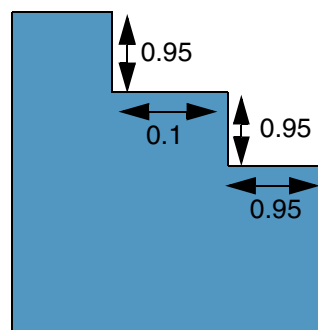
a) FAIL. The number of consecutive edges shorter than 1.0 is 3 (>2).



b) FAIL. The number of consecutive edges shorter than 1.0 is 3 (>2).



c) PASS. The number of consecutive edges shorter than 1.0 is 2.

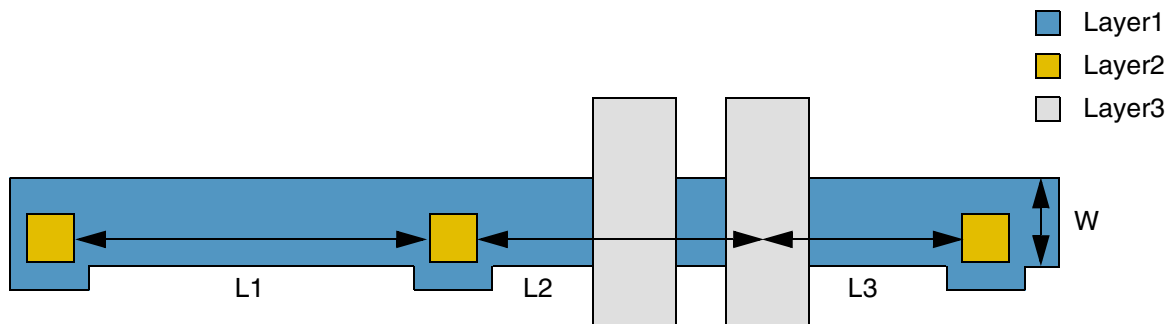


d) PASS. The number of consecutive edges shorter than 1.0 is 2.

maxPolyLength

```
orderedSpacings (
    ( maxPolyLength tx_layer1 tx_layer2 tx_layer3
      ['width f_width]
      f_length
    )
) ;orderedSpacings
```

Specifies the maximum allowed distance between *layer2* edges, along *layer1*, or from any point on *layer3* over *layer1*, along *layer1*, when the width of the *layer1* shape is less than *width*.



Values

<i>tx_layer1</i>	The first (poly routing) layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second (cut) layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer3</i>	The third (gate forming, such as active or oxide) layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The maximum allowed distance between <i>layer2</i> edges, along <i>layer1</i> , or from any point on <i>layer3</i> over <i>layer1</i> , along <i>layer1</i> .

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

`'width  $f\_width$`

The constraint applies only if the width of the *layer1* shape is less than this value.

Example

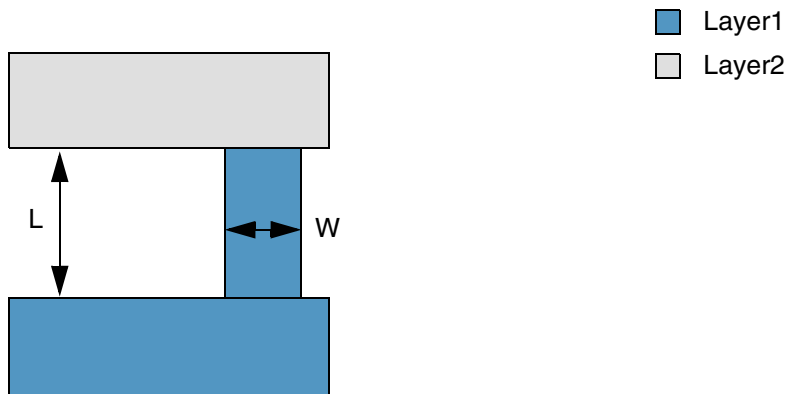
The maximum allowed distance between Cont edges, along Poly, or from any point on Oxide over Poly, along Poly, must be 0.5 when the width of the Poly shape is less than 0.2.

```
orderedSpacings (
  ( maxPolyLength "Poly" "Cont" "Oxide"
    'width 0.2
    0.5
    'ref "MAXPOLY"
    'description "maxPolyLength"
  )
) ;orderedSpacings
```

maxTouchingDirectionLength

```
spacingTables (
  ( maxTouchingDirectionLength tx_layer1 tx_layer2
    (( "width" nil nil )
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the maximum length of a *layer1* shape with width less than or equal to the given width, in the direction normal to the *layer1* edge that is touched (abutted or overlapped) by a shape on *layer2*.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

g_table

The format of the *table* row is as follows:

(*f_width f_length*)

where,

- *f_width* is the width of the *layer1* shape.
- *f_length* is the maximum length of the *layer1* shape when the width of the *layer1* shape is less than or equal to the index value.

Type: A 1-D table specifying floating point width and length values

Parameters

f_default

The length value to be used when no table entry applies.

Example

The maximum length of the Metal1 shape is indexed on the width of the shape, and the direction in which length is measured is determined by the Metal2 shape that touches the Metal1 shape.

```
spacingTables(  
  ( maxTouchingDirectionLength "Metal1" "Metal2"  
    (( "width" nil nil )  
      0.4  
    )  
    (  
      0.5    4.5  
      0.6    7.5  
      0.7    4.3  
    )  
  )  
) ;spacingTables
```

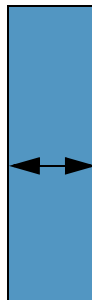
Virtuoso Technology Data Constraints Reference

Length and Width Constraints

maxWidth

```
spacings(  
  ( maxWidth tx_layer  
    ['vertical | 'horizontal]  
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN) ]  
      |  
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN) ]  
    ]  
    f_width  
  )  
); spacings
```

Specifies the maximum width for a shape on the specified layer.



■ Metal1

Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_width

The width of the shape must be less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

'horizontal' | 'vertical'

(Virtuoso Advanced Node for Layout Only) The direction in which width is measured. If direction is not specified, width is measured in any direction.

Note: This parameter can be used to model the width of a shape in a direction perpendicular to the routing direction specified for the layer.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Example

The maximum width of shapes on Metal3 must be 2.5.

```
spacings(
  ( maxWidth "Metal3" 2.5 )
) ;spacings
```

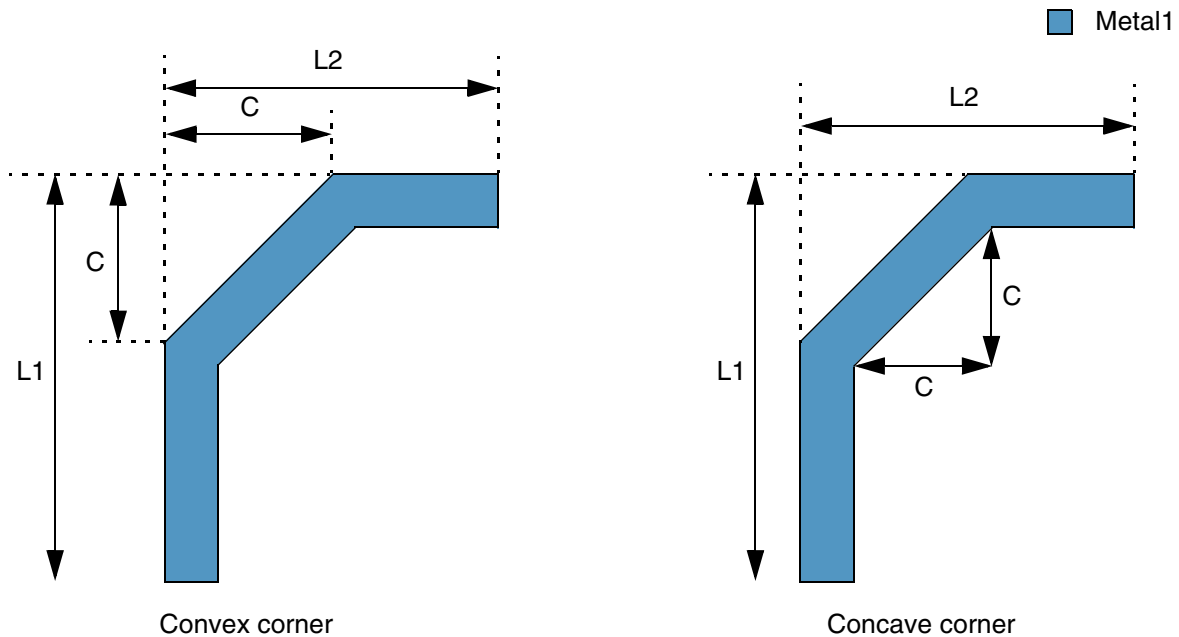
minChamferLength

```
spacingTables (
  ( minChamferLength tx_layer
    (( "length" nil nil )
      ['convex | 'concave]
    )
    g_table
  )
) ;spacingTables
```

Specifies required length of chamfer, indexed by adjacent edge lengths.

In some advanced node processes, 90-degree corners are not allowed on last layer metal. This constraint models this rule.

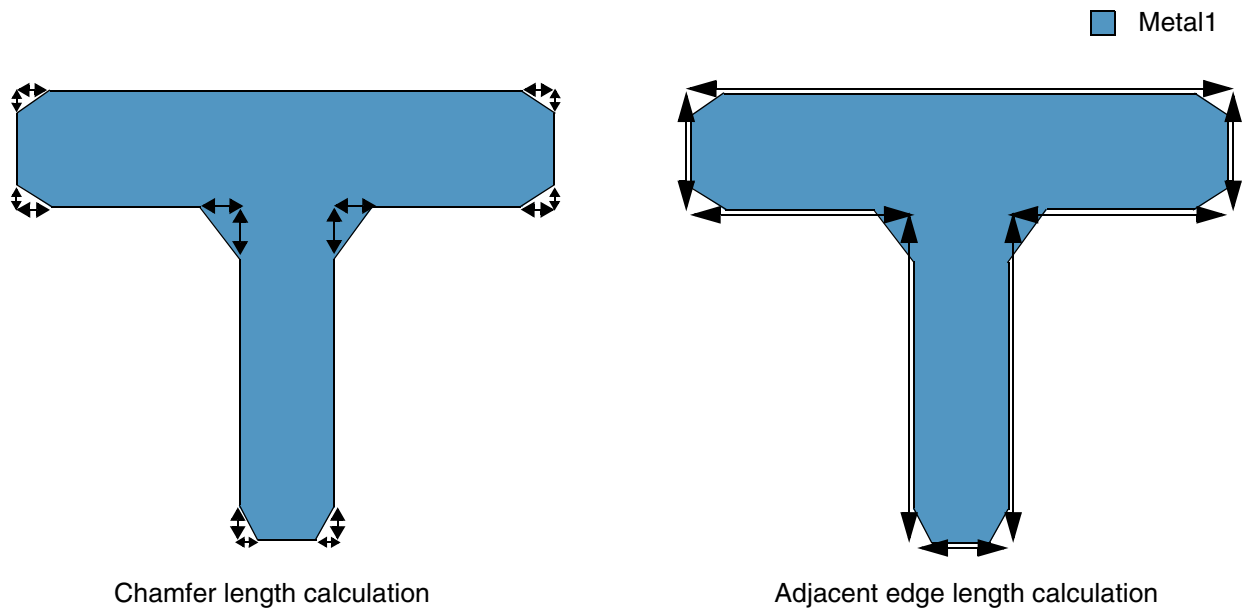
The figure below illustrates chamfer and adjacent edge lengths are calculated for convex and concave corners.



The figure below illustrates how chamfer and adjacent edge lengths are calculated for shapes with more than one chamfer.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

"length" nil nil

This identifies the index for *table*.

g_table

The format of the *table* row is as follows:

(*f_adjLength* *f_length*)

where,

- *f_adjLength* is the adjacent edge length.
- *f_length* is the required chamfer length.

Type: A 1-D table specifying floating point length values

Parameters

'convex' | 'concave' The corner type to which the constraint applies.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Examples

- Example 1: minChamferLength with one corner
- Example 2: minChamferLength with multiple corners

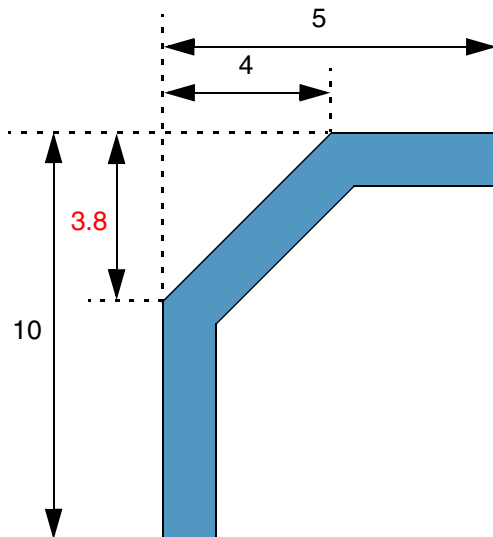
Example 1: minChamferLength with one corner

The chamfer length for concave and convex corners must be at least:

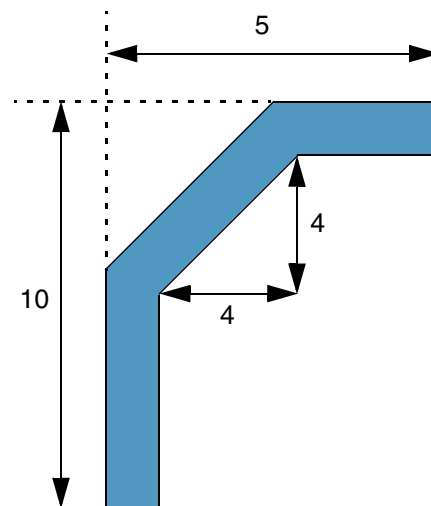
- 3.0 when adjacent edge length is 5
- 4.0 when adjacent edge length is 10
- 5.0 when adjacent edge length is 15

```
spacingTables (
  ( minChamferLength "Metal1"
    (( "length" nil nil ))
    (
      5    3.0
      10   4.0
      15   5.0
    )
  )
) ;spacingTables
```

■ Metal1



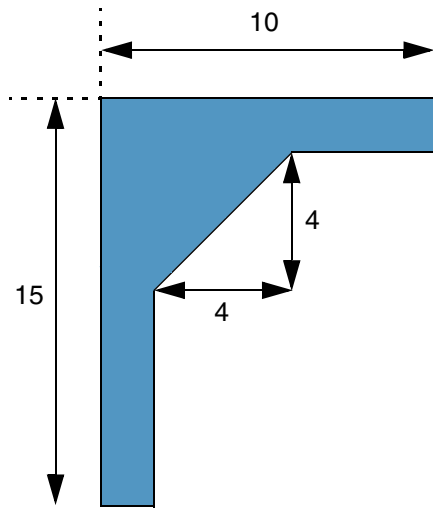
a) FAIL. When adjacent edge width is 10, the chamfer length must be at least 4.



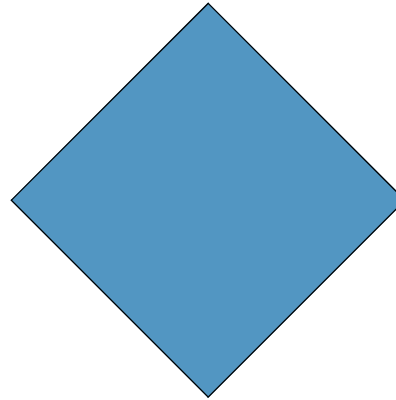
b) PASS. Chamfer length is satisfied for adjacent edge widths 5 and 10.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints



c) FAIL. There is no chamfering for the convex corner.



d) FAIL. Right-angled corners are not allowed, even if formed by diagonal edges.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

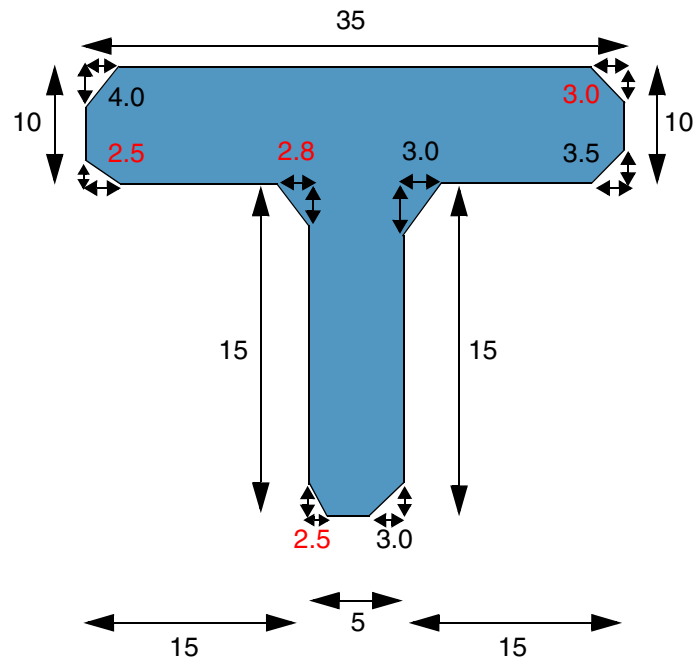
Example 2: *minChamferLength* with multiple corners

The chamfer length for concave and convex corners must be at least:

- 1.0 when adjacent edge length is 5
- 2.0 when adjacent edge length is 10
- 3.0 when adjacent edge length is 15
- 4.0 when adjacent edge length is 35

```
spacingTables(  
  ( minChamferLength "Metal1"  
    (( "length" nil nil ))  
    (  
      5    1.0  
      10   2.0  
      15   3.0  
      35   4.0  
    )  
  )  
) ;spacingTables
```

■ Metal1

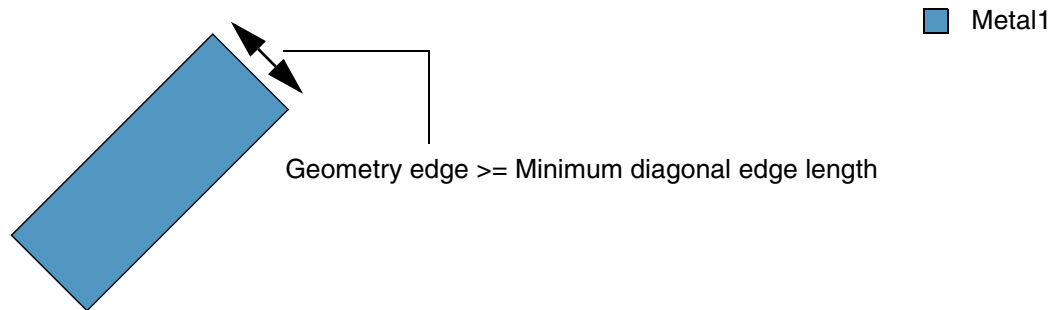


FAIL. Top two corners need a chamfer length of at least 4.0 because the adjacent edge length is 35, while the remaining corners need a chamfer length of at least 3.0 because the maximum adjacent edge length is 15.

minDiagonalEdgeLength

```
spacings (
    ( minDiagonalEdgeLength tx_layer {f_length | (f_lengthSum f_length)} )
) ;spacings
```

Sets the minimum edge length for a diagonal shape on the specified layer.



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_length | (f_lengthSum f_length)

■ *f_edgeLength*

The length of the diagonal edge must be greater than or equal to this value.

■ *(f_lengthSum f_edgeLength)*

The constraint applies only if the sum of lengths of the consecutive edges that are shorter than *edgeLength* is less than *lengthSum*.

Parameters

None

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Example

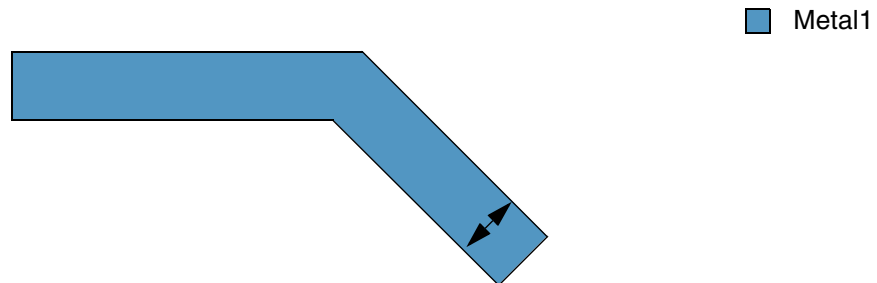
The minimum diagonal edge length must be 1.5 on Metal1 and equal to the value of the technology parameter `diaglength1` on Metal2.

```
spacings (
  ( minDiagonalEdgeLength "Metal1" 1.5 )
  ( minDiagonalEdgeLength "Metal2" techParam("diaglength1") )
) ;spacings
```

minDiagonalWidth

```
spacings(  
    ( minDiagonalWidth tx_layer f_width )  
); spacings
```

Sets the minimum width of diagonal shapes on the specified layer.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_width</i>	The minimum width.

Parameters

None

Example

The minimum width of diagonal shapes must be 1.0 on Metal1 and equal to the value of the technology parameter `minddiagwidth1` on Metal2.

```
spacings(  
    ( minDiagonalWidth "Metal1" 1.0 )  
    ( minDiagonalWidth "Metal2" techParam("minddiagwidth1") )  
); spacings
```

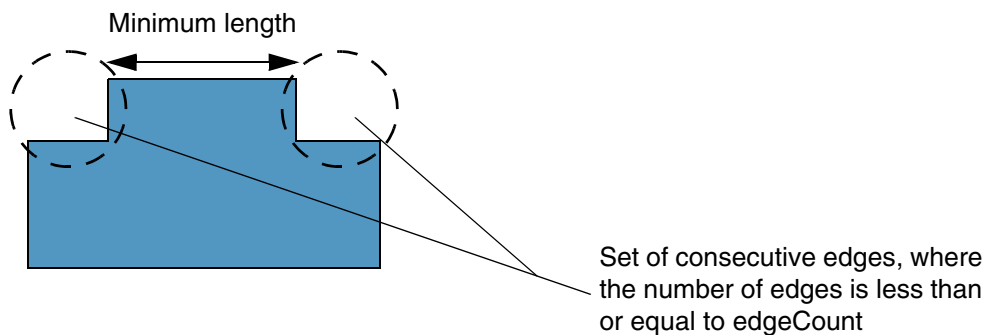
minEdgeAdjacentDistance

```
spacings (
    ( minEdgeAdjacentDistance tx_layer
      'length f_length
      'edgeCount x_edgeCount
      ['exceptSameCorner]
      f_distance
    )
) ; spacings
```

Specifies the minimum distance along the edge between two sets of connected edges. Each connected edge must be shorter than *length* and the number of connected edges must be less than or equal to *edgeCount*.

edgeCount = 2

■ Metal1



This constraint is usually applied in combination with the [maxNumMinEdges](#) constraint.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_distance</i>	The distance between two sets of connected edges must be greater than or equal to this value.

Parameters

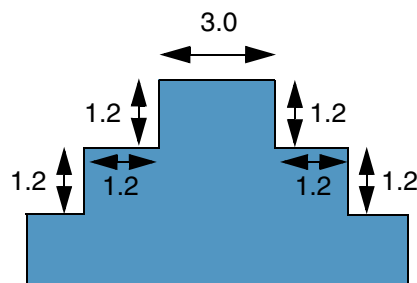
'length <i>f_length</i>	Each connected edge must have length less than this value.
'edgeCount <i>x_edgeCount</i>	The number of connected edges shorter than <i>length</i> must be less than or equal to this value.
'exceptSameCorner	The constraint does not apply to an edge that has the same type of 90-degree corner (convex or concave) at each end.

Example

The distance between two sets of connected edges, each with length less than 1.5, must be greater than or equal to 3.0, provided the number of connected edges in each set is less than or equal to 4.

```
spacings(  
  ( maxNumMinEdges "Metal1" (1.5 4) )  
  ( minEdgeAdjacentDistance "Metal1"  
    'length 1.5  
    'edgeCount 4  
    3.0  
  )  
) ;spacings
```

■ Metal1



PASS. The number of connected edges with length 1.2 (<1.5) is 3 (<4) and the distance between the two sets of consecutive edges is 3.0, which satisfies the `minEdgeAdjacentDistance` constraint.

minEdgeAdjacentLength

```

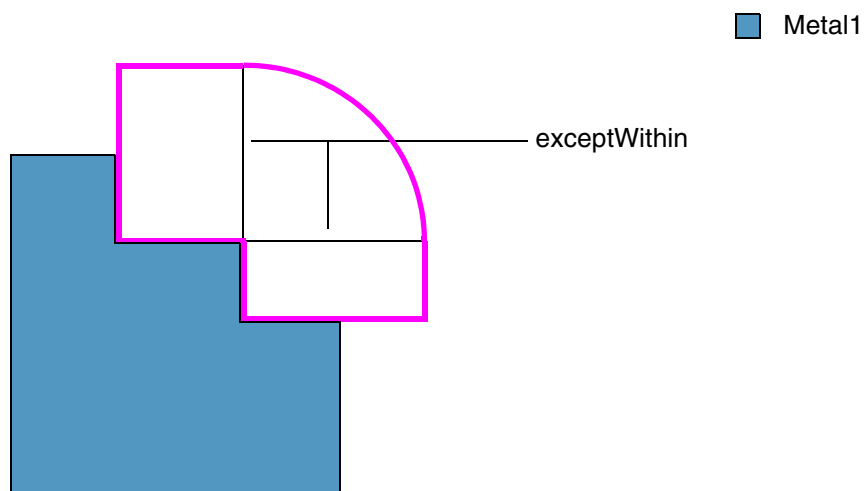
spacings (
  ( minEdgeAdjacentLength tx_layer
    'maxLength f_maxLength
    ['edgeCount x_maxEdges]
    ['convexCorner ['exceptWithin f_within]
      | 'concaveCorner
      | 'threeConcaveCorners ['width f_width]
    ]
    f_length
  )
  ( minEdgeAdjacentLength tx_layer
    'maxLength f_maxLength
    ['edgeCount x_maxEdges]
    (f_length1 f_length2)
  )
) ;spacings

```

Specifies the minimum lengths allowed for edges adjacent to a short edge that is less than *maxLength* long.

Foundries require control over the minimum length of edges adjacent to a short edge of a polygon. Some foundries require control over the minimum length of each edge adjacent to a short edge.

When 'exceptWithin is specified, the constraint does not apply if a neighboring shape is present inside the region highlighted in pink, as shown below.



Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The lengths of both adjacent edges must be greater than or equal to this value.
<i>f_length1 f_length2</i>	The length of one adjacent edge must be greater than or equal to <i>length1</i> and the length of the other adjacent edge must be greater than or equal to <i>length2</i> . The two lengths can be identical.

Parameters

'maxLength <i>f_maxLength</i>	The constraint applies only if the length of a short edge is less than this value.
'edgeCount <i>x_maxEdges</i>	Specifies that up to <i>edgeCount</i> consecutive edges that are less than <i>maxLength</i> in length are allowed, but more than <i>edgeCount</i> in a row is a violation.
'convexCorner	If one edge of a convex corner that lies between two concave corners is less than <i>length</i> (the constraint value), the other edge must be greater than or equal to <i>maxLength</i> . The constraint does not apply if a convex corner is not found between two concave corners.
'exceptWithin <i>f_within</i>	The constraint does not apply if a neighboring shape is present (on the same layer as the polygon) at a distance less than this value from the convex corner or the two edges that form the convex corner. The distance is measured in Euclidean distance.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

'concaveCorner	(Virtuoso Advanced Node for Layout Standard) If one edge of a concave corner that lies between two convex corners is less than <i>maxLength</i> , the other edge must be greater than or equal to <i>length</i> (the constraint value). The constraint does not apply if a concave corner is not found between two convex corners.
'threeConcaveCorners	(Virtuoso Advanced Node for Layout Only) The constraint applies only if there exist three consecutive concave corners with two convex corners between them. The length of the edges that form the middle concave corner must be greater than or equal to <i>length</i> (the constraint value). Type: Boolean
'width <i>f_width</i>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the widths of the shapes that are connected to form the middle concave corner are less than or equal to this value.

Examples

- Example 1: minEdgeAdjacentLength with maxLength and edgeCount
- Example 2: minEdgeAdjacentLength with maxLength, edgeCount, and convexCorner
- Example 3: minEdgeAdjacentLength with maxLength, edgeCount, threeConcaveCorners, and width

Virtuoso Technology Data Constraints Reference

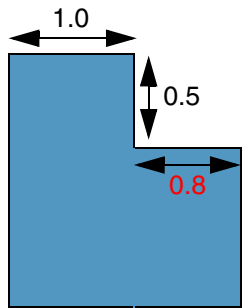
Length and Width Constraints

Example 1: *minEdgeAdjacentLength* with *maxLength* and *edgeCount*

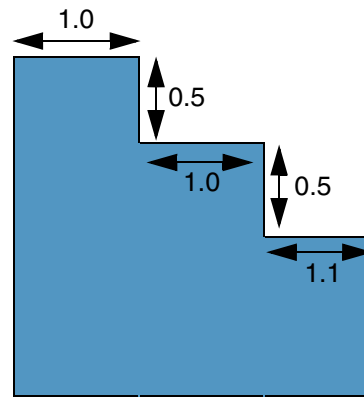
If a polygon has at most one short edge less than 0.6 long, the edges adjacent to the short edge must be at least 1.0 long.

```
spacings(  
  ( minEdgeAdjacentLength "Metal1"  
    'maxLength 0.6  
    'edgeCount 1  
    1.0  
  )  
); spacings
```

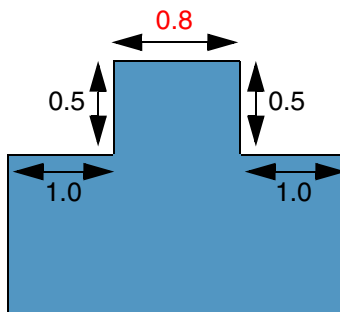
■ Metal1



a) FAIL. The vertical edge that is 0.5 (<0.6) long, has an adjacent edge that is only 0.8 long (<1.0).



b) PASS. Both adjacent edges of all edges that are 0.5 (<0.6) long are 1.0 long.



c) FAIL. Each vertical edge that is 0.5 (<0.6) long, has an adjacent edge that is only 0.8 long (<1.0).

Virtuoso Technology Data Constraints Reference

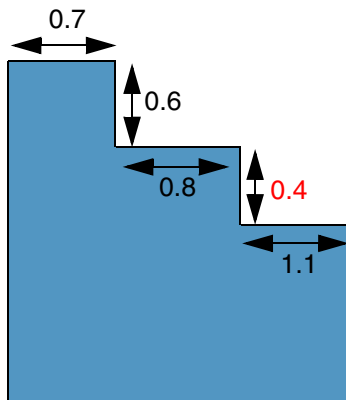
Length and Width Constraints

Example 2: *minEdgeAdjacentLength* with *maxLength*, *edgeCount*, and *convexCorner*

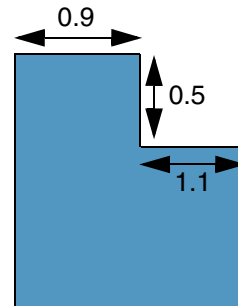
The convex corner formed by 0.4 and 0.8 long edges lies between two concave corners. If one edge of the convex corner is less than 1.0, then the other edge must be greater than or equal to 0.6, and if both edges are less than 1.0, then both edges must also be greater than or equal to 0.6.

```
spacings (
  ( minEdgeAdjacentLength "Metal1"
    'maxLength 0.6
    'edgeCount 1
    'convexCorner
    1.0
  )
) ;spacings
```

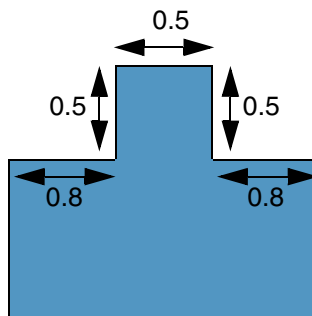
■ Metal1



a) FAIL. Here, both edges are less than 1.0, so both edges must also be greater than or equal to 0.6. When 0.4 is less than 1.0, 0.8 is greater than 0.6. However, when 0.8 is less than 1.0, 0.4 is not greater than 0.6.



b) The constraint does not apply because the convex corner formed by the 0.5 and 0.9 long edges does not lie between two concave corners.



c) The constraint does not apply because the convex corners do not lie between concave corners.

Virtuoso Technology Data Constraints Reference

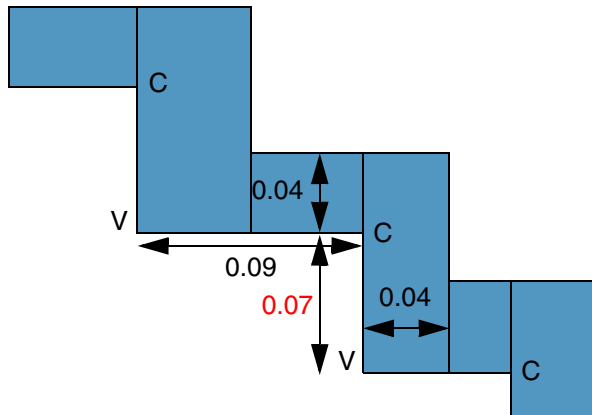
Length and Width Constraints

Example 3: *minEdgeAdjacentLength* with *maxLength*, *edgeCount*, *threeConcaveCorners*, and *width*

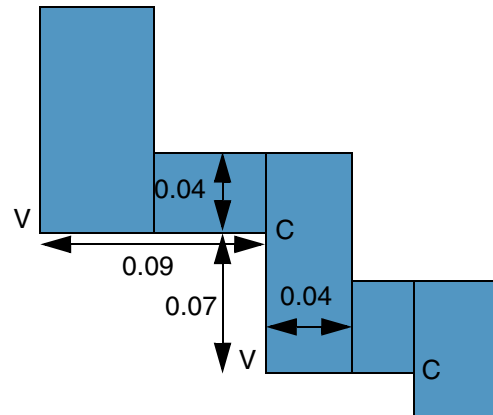
If there exist three consecutive concave (C) corners with two convex (V) corners in between, then the length of both edges that form the middle concave corner must be greater than or equal to 0.08, provided the shapes that are connected to form the middle concave corner are less than or equal to 0.04 wide.

```
spacings (
  ( minEdgeAdjacentLength "Metal1"
    'maxLength 0.1
    'edgeCount 1
    'threeConcaveCorners
    'width 0.04
    0.08
  )
) ;spacings
```

■ Metal1



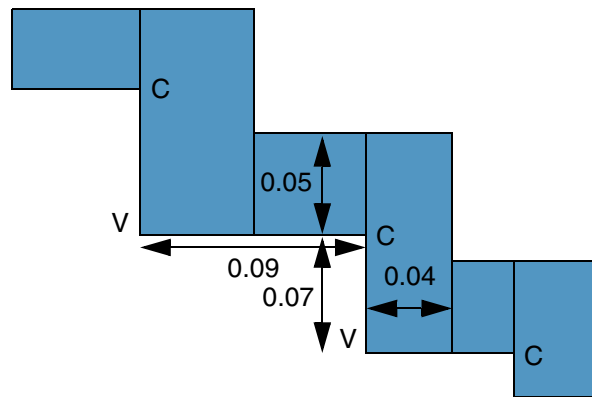
a) FAIL. The shapes that form the middle concave corner are 0.04 wide, but the length of the vertical edge that forms the middle concave corner is only 0.07 (<0.08).



b) The constraint does not apply because there exist only two concave corners.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints



c) The constraint does not apply because the width of one of the shapes that forms the concave corner is 0.05 (>0.04). A violation would occur if 'width' was not specified.

minEndOfLineAdjacentToStep

```

spacings (
    ( minEndOfLineAdjacentToStep tx_layer
      'maxLength f_maxLength
      ['exceptAdjacentLength f_exceptAdjacentLength
        | 'adjacentLength f_adjacentLength
        | 'noBetweenEol
      ]
      ['concaveCorner]
      f_length
    )
) ;spacings

```

Specifies the minimum length for an end-of-line edge that is adjacent to an edge with length less than *maxLength*.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The minimum length.

Parameters

'maxLength *f_maxLength*

The constraint applies only if the end-of-line edge has an adjacent edge that is shorter than this value (minimum step edge).

'exceptAdjacentLength *f_exceptAdjacentLength*

The constraint does not apply if the other adjacent edge of the minimum step edge has length greater than or equal to this value.

'adjacentLength *f_adjacentLength*

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the other adjacent edge of the minimum step edge has length greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

'noBetweenEol	An end-of-line edge with length less than <i>length</i> (the constraint value) cannot have two adjacent edges with length less than <i>maxLength</i> . Type: Boolean
'concaveCorner	(Virtuoso Advanced Node for Layout Standard) The constraint applies only if both edges adjacent to an end-of-line edge form a concave corner at the other end. Type: Boolean

Examples

- Example 1: minEndOfLineAdjacentToStep with maxLength and adjacentLength
- Example 2: minEndOfLineAdjacentToStep with maxLength and exceptAdjacentLength
- Example 3: minEndOfLineAdjacentToStep with maxLength and noBetweenEol
- Example 4: minEndOfLineAdjacentToStep with maxLength and concaveCorner

Virtuoso Technology Data Constraints Reference

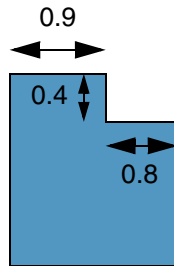
Length and Width Constraints

Example 1: *minEndOfLineAdjacentToStep* with *maxLength* and *adjacentLength*

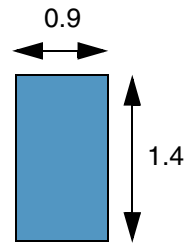
The length of an end-of-line edge must be greater than or equal to 1.0 if it is adjacent to an edge with length less than 1.5, which, in turn, forms a concave corner with an edge with length greater than or equal to 0.7 at the other end.

```
spacings (  
  ( minEndOfLineAdjacentToStep "Metal1"  
    'maxLength 1.5  
    'adjacentLength 0.7  
    1.0  
  )  
) ;spacings
```

■ Metal1



a) FAIL. One edge adjacent to the edge with length 0.4 (<1.5) is an end-of-line edge and the length of the other adjacent edge is 0.8 (>0.7), but the length of the end-of-line edge is only 0.9 (<1.0).



b) The constraint does not apply because the vertical minimum step edge with length 1.4 (<1.5) does not form a concave corner at the other end.

Virtuoso Technology Data Constraints Reference

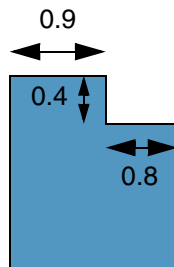
Length and Width Constraints

Example 2: minEndOfLineAdjacentToStep with maxLength and exceptAdjacentLength

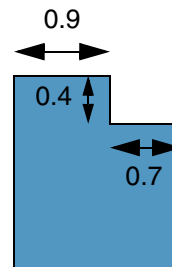
The length of an end-of-line edge must be greater than or equal to 1.0 if it is adjacent to an edge with length less than 1.5. The constraint does not apply if the minimum step edge forms a concave corner with an edge with length greater than or equal to 0.8 at the other end.

```
spacings (
  ( minEndOfLineAdjacentToStep "Metal1"
    'maxLength 1.5
    'exceptAdjacentLength 0.8
    1.0
  )
) ;spacings
```

■ Metal1



a) The constraint does not apply. One edge adjacent to the edge with length 0.4 (<1.5) is an end-of-line edge and the length of the other adjacent edge is 0.8 (the constraint does not apply if this length is ≥ 0.8).



b) FAIL. One edge adjacent to the edge with length 0.4 (<1.5) is an end-of-line edge and the length of the other adjacent edge is 0.7 (<0.8), but the length of the end-of-line edge is only 0.9 (<1.0).

Virtuoso Technology Data Constraints Reference

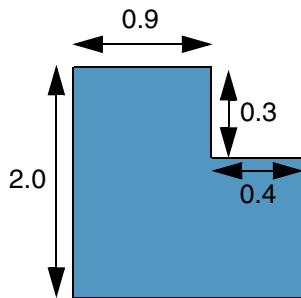
Length and Width Constraints

Example 3: minEndOfLineAdjacentToStep with maxLength and noBetweenEol

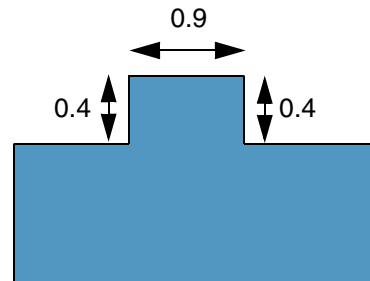
The length of an end-of-line edge must be greater than or equal to 1.0 if it has two adjacent edges with length less than 1.5.

```
spacings (
  ( minEndOfLineAdjacentToStep "Metal1"
    'maxLength 1.5
    'noBetweenEol
    1.0
  )
) ;spacings
```

■ Metal1



a) The constraint does not apply because the end-of-line edge has only one adjacent edge with length less than 1.5.



b) FAIL. The end-of-line edge has two adjacent edges with length 0.4 (< 1.5), but the length of the end-of-line edge is only 0.9 (< 1.0).

Virtuoso Technology Data Constraints Reference

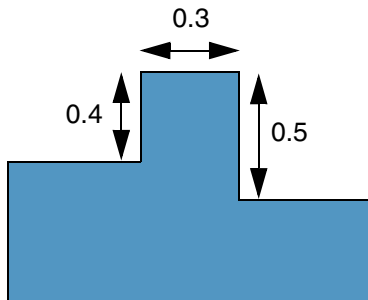
Length and Width Constraints

Example 4: minEndOfLineAdjacentToStep with maxLength and concaveCorner

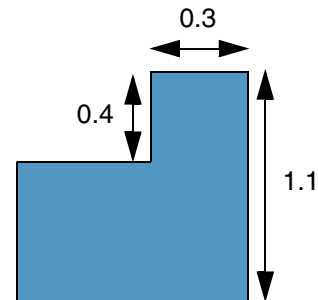
The length of an end-of-line edge must be greater than or equal to 1.0 if it has adjacent edges with length less than 1.5 and these adjacent edges form concave corners at their ends.

```
spacings (
  ( minEndOfLineAdjacentToStep "Metal1"
    'maxLength 1.5
    'concaveCorner
    1.0
  )
) ;spacings
```

■ Metal1



a) The end-of-line edge has two adjacent edges with lengths 0.4 and 0.5 (<1.5) and these adjacent edges form concave corners at their other ends, but the length of the end-of-line edge is only 0.3 (<1.0).



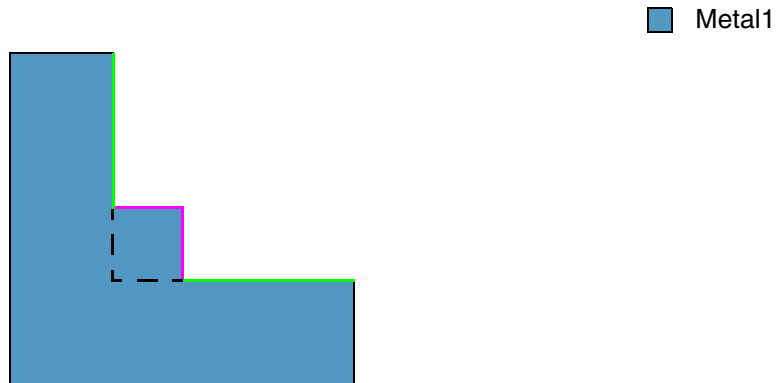
b) The constraint does not apply because the adjacent edge with length 1.1 (<1.5) does not form a concave corner at its other end.

minInsideCornerEdgeLength

```
spacings (
    ( minInsideCornerEdgeLength tx_layer
      {f_edgeLength | (f_lengthSum f_edgeLength)}
    )
) ;spacings
```

Sets the minimum length of an edge that is in a concave corner of a shape.

The figure below illustrates a concave shape because the two green edges that are greater than or equal to *edgeLength* long abut consecutive edges (in pink) shorter than the minimum edge length.



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_edgeLength

The edges in a concave corner must be greater than or equal to this value.

Parameters

f_lengthSum

The constraint does not apply if the sum of the consecutive edges that are shorter than *edgeLength* is less than this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Example

The minimum inside corner edge length on Poly1 and Metal2 must be 1.5. The constraint does not apply on Metal2 if the sum of lengths of the consecutive edges that are shorter than 1.5 is less than 5.0.

```
spacings (
  ( minInsideCornerEdgeLength "Poly1" 1.5 )
  ( minInsideCornerEdgeLength "Metal2" (5.0 1.5) )
) ;spacings
```

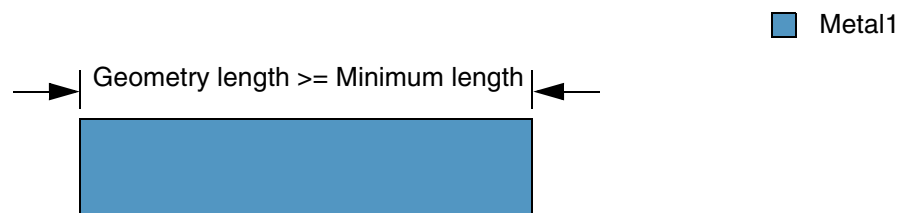

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

minLength

```
spacings (
    ( minLength tx_layer
      ['width f_width]
      ['exceptViaLayers (tx_viaLayer1 tx_viaLayer2 ... tx_viaLayerN) ]
      f_length )
) ;spacings
```

Determines the minimum orthogonal length for a shape on the specified layer.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The length of the shape must be greater than or equal to this value.

Parameters

<i>'width f_width</i>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when the width of the shape is less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

```
'exceptViaLayers tx_viaLayer1 tx_viaLayer2 ...tx_viaLayerN
```

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when there is no cut on the specified via layers.

Type: String (layer name) or Integer (layer number); Float

Example

The minimum orthogonal length for a shape on Metal1 must be 0.3.

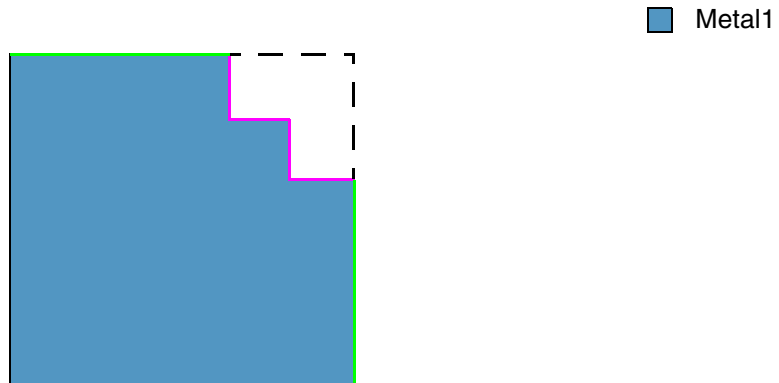
```
spacings(  
  ( minLength gate 0.3 )  
) ;spacings
```

minOutsideCornerEdgeLength

```
spacings (
    ( minOutsideCornerEdgeLength tx_layer
      {f_edgeLength | (f_lengthSum f_edgeLength)}
    )
) ;spacings
```

Sets the minimum length of an edge in a convex corner of a shape.

The figure below illustrates a convex shape because the two green edges that are greater than or equal to *edgeLength* long abut consecutive edges (in pink) shorter than the minimum edge length.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_edgeLength</i>	The edges in a convex corner must be greater than or equal to this value.

Parameters

<i>f_lengthSum</i>	The constraint does not apply if the sum of the consecutive edges that are shorter than <i>edgeLength</i> is less than this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Example

The minimum outside corner edge length on Poly1 and Metal2 must be 1.5. The constraint does not apply on Metal2 if the sum of lengths of the consecutive edges that are shorter than 1.5 is less than 7.0.

```
spacings (
  ( minOutsideCornerEdgeLength "Poly1" 1.5 )
  ( minOutsideCornerEdgeLength "Metal2" (7.0 1.5) )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

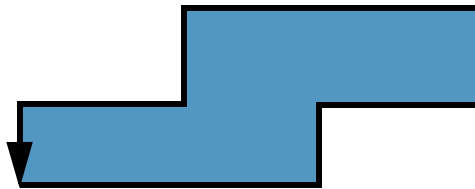
Length and Width Constraints

minPerimeter

```
spacings(  
    ( minPerimeter tx_layer f_perimeter )  
); spacings
```

Specifies the minimum perimeter of a shape on the given layer.

■ Metal1



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_perimeter

The perimeter of a shape must be greater than or equal to this value.

Parameters

None

minProtrusionWidthLength

```

spacings (
    ( minProtrusionWidthLength tx_layer
      'widthLengthTable ((f_wideWidth f_length) ...)
      ['enclosedWidth f_enclWidth
       | ['exceptCutTable (l_distance) ['layer tx_cutLayer]
      ]
      f_width
    )
) ;spacings

```

Specifies the minimum width and length for a protrusion on a wide wire. The length of the protrusion varies based on the width of the wide wire.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_width</i>	The width of the protrusion must be greater than or equal to this value.

Parameters

<i>'widthLengthTable (f_wideWidth f_length)</i>	If the width of the wide wire on which the protrusion exists is greater than or equal to <i>wideWidth</i> , the length of the protrusion must be greater than or equal to <i>length</i> .
<i>'enclosedWidth f_enclWidth</i>	(Virtuoso Advanced Node for Layout Standard) The width of the protrusion, excluding any portion of the wide wire, must be greater than or equal to this value.
<i>'exceptCutTable (l_distance)</i>	The constraint does not apply if there exists a via cut on layer <i>cutLayer</i> at a distance less than this value from the wide wire.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

```
'layer tx_cutLayer
```

The constraint does not apply if the via cut that is at a distance less than *distance* from the wide wire is present on this layer.

Type: String (layer name) or Integer (layer number)

Examples

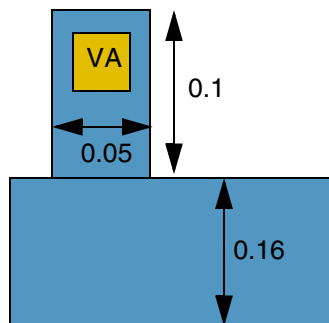
- Example 1: minProtrusionWidthLength with widthLengthTable, exceptCutTable, and layer
- Example 2: minProtrusionWidthLength with widthLengthTable and enclosedWidth

Example 1: minProtrusionWidthLength with widthLengthTable, exceptCutTable, and layer

A protrusion on a wide Metal1 wire with width greater than or equal to 0.11 must be at least 0.05 wide and at least 0.12 long. This rule does not apply if a Via1 via cut is present at a distance less than 0.1 from the wide wire.

```
spacings(  
  ( minProtrusionWidthLength "Metal1"  
    'widthLengthTable (0.11 0.12)  
    'exceptCutTable (0.1) 'layer "Via1"  
    0.05  
  )  
) ;spacings
```

■ Via1
■ Metal1



The constraint does not apply because the via cut on layer Via1 is at a distance less than 0.1 from the wide wire.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

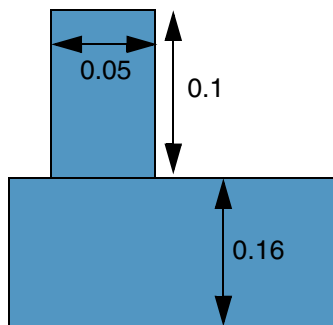
Example 2: *minProtrusionWidthLength* with *widthLengthTable* and *enclosedWidth*

The following conditions must be met:

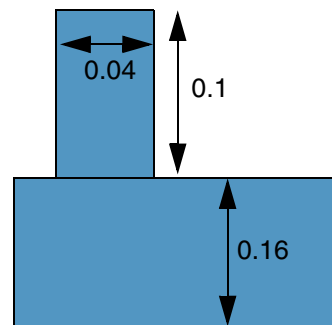
- The width of a protrusion on a wide Metal1 wire must be at least 0.05.
- If the width of the wide wire is greater than or equal to 0.11, the length of the protrusion must be at least 0.12.
- The width of the protrusion after excluding any width of the wide wire must be equal to 0.02.

```
spacings (  
  ( minProtrusionWidthLength "Metal1"  
    'widthLengthTable (0.11 0.12)  
    'enclosedWidth 0.02  
    0.05  
  )  
) ;spacings
```

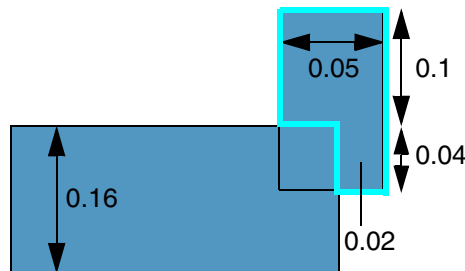
■ Metal1



a) FAIL. The width of the protrusion is 0.05, but the length of the protrusion is only 0.1 (<0.12).



b) The constraint does not apply because the width of the protrusion is less than 0.05.



c) PASS. The width of the protrusion is 0.05 and its length 0.14 (>0.12). Its width excluding the width of the wide wire is 0.02.

Virtuoso Technology Data Constraints Reference

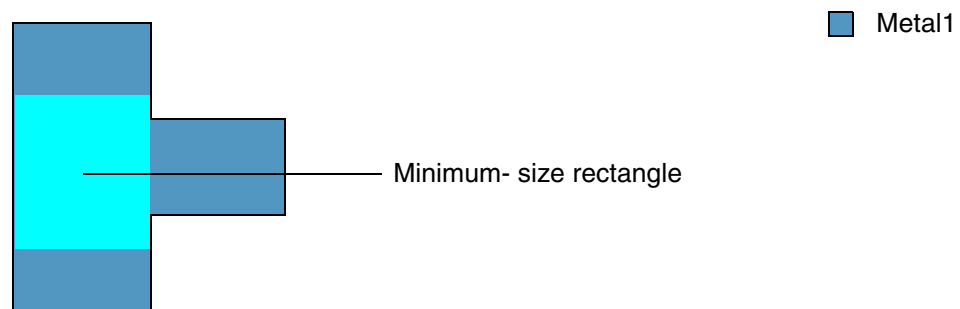
Length and Width Constraints

minSize

```
spacings(  
    ( minSize tx_layer  
      ['rectOnly']  
      ((f_width f_length) ...)  
    )  
); spacings
```

Defines the minimum rectangle size that must fit inside all polygons on the specified layer. If you specify multiple minimum rectangle sizes, the constraint is satisfied if any one of the rectangles fits inside all polygons on the layer.

Note: If both `minArea` and `minSize` constraints are set on a layer, both constraints are satisfied if either constraint is met.



Values

`tx_layer`

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

`((f_width f_length) ...)`

The width and length of the rectangle. The constraint value should be a list of lists, but a simple list can also be specified.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

`'rectOnly` The constraint applies only to rectangular shapes.

Examples

- [Example 1: minSize](#)
- [Example 2: minSize with rectOnly](#)

Example 1: minSize

A 0.1x0.2 rectangle must fit inside every rectangular shape on Metal1.

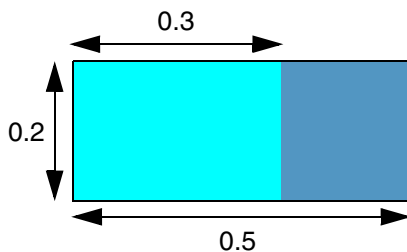
```
spacings(  
  ( minSize "Metal1"  
    'rectOnly  
    (0.1 0.2)  
  )  
); spacings
```

Example 2: minSize with rectOnly

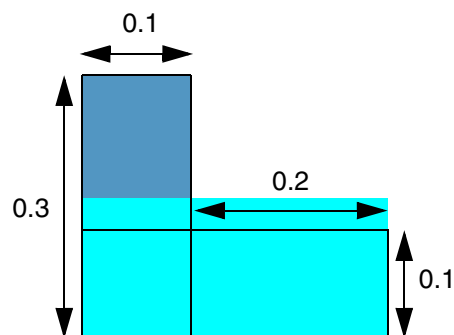
A 0.1x0.2 or a 0.2x0.3 or a 0.3x0.4 rectangle must fit inside every rectangular shape on Metal1.

```
spacings(  
  ( minSize "Metal1"  
    'rectOnly  
    ((0.1 0.2) (0.2 0.3) (0.3 0.4))  
  )  
); spacings
```

■ Metal1



a) PASS. The rectangle with a minimum size of 0.2x0.3 fits inside the rectangle on Metal1.



b) The constraint does not apply. Because `'rectOnly` is specified, the constraint applies only to rectangular shapes.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

minStepEdgeLength

```
spacings (
  ( minStepEdgeLength tx_layer
    ['any | 'horizontalEdge | 'verticalEdge]
    ['allCorner | 'concaveCorner | 'convexCorner | 'mixedCorner]
    ['extendToAlign (g_width g_spacing)
      ['extendOnly | 'alignOnly]]
    ['width g_width ['adjEolSpacingRange g_eolSpaceRange]]
    ['protrusionEdge]
    ['jogEdge]
    ['adjJogEdge]
    ['adjNotchEdge]
    ['exceptExactLength ((f_exactLength f_adjLength) ...)]
    ['exceptExactBothLength ((f_exactLength f_exactAdjLength) ...)]
    ['exceptRectangle]
    ['fillerOnly]
    {f_edgeLength | (f_lengthSum f_edgeLength)}
  )
) ;spacings
```

Specifies the minimum edge length for a shape on the specified layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_edgeLength</i>	The length of each edge must be greater than or equal to this value.

Parameters

'any 'horizontalEdge 'verticalEdge	(Virtuoso Advanced Node for Layout Standard) The constraint applies only to the edges in the specified direction. The default is 'any. Type: Boolean
'allCorner 'concaveCorner 'convexCorner 'mixedCorner	

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

(Virtuoso Advanced Node for Layout Standard) The constraint applies to an edge between two concave corners, two convex corners, or a concave and a convex corner. The default is 'allCorner.

Type: Boolean

`'extendToAlign (g_width g_spacing) ['extendOnly | 'alignOnly]`

(Virtuoso Advanced Node for Layout Only) Specifies that if shapes on the upper and lower edges within a cell row have width equal to *g_width*, the shapes are extended and aligned. In addition, any gap on the same edge that is less than or equal to *g_spacing* is filled and merged while calculating the length.

Optionally, you can specify either of the following additional parameters:

- `extendOnly`: Only the extension operation is performed.
- `alignOnly`: Only the alignment operation is performed.

`'width g_width`

(Virtuoso Advanced Node for Layout Only) The constraint applies only on shapes with width equal to the given value.

`'adjEolSpacingRange g_eolSpaceRange`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies to an edge if any one of its adjacent edges is an end-of-line edge meeting the width parameter and this adjacent edge has another perfectly aligned end-to end-shape at a spacing specified by this parameter.

`'protrusionEdge`

(Virtuoso Advanced Node for Layout Only) The constraint applies only to a protrusion edge.

`'jogEdge`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to a jog edge (Z shape).

`'adjJogEdge`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to an edge adjacent to a jog edge.

`'adjNotchEdge`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to an edge adjacent to a notch edge.

The `adjJogEdge` and `adjNotchEdge` parameters can be used together in a rule in case the rule is to be applied on an edge that is adjacent to a jog as well as a notch.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

'exceptExactLength ((*f_exactLength* *f_adjLength*) ...)

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if an edge with length equal to *exactLength* has an adjacent edge with length less than or equal to *adjLength*.

'exceptExactBothLength ((*f_exactLength* *f_exactAdjLength*) ...)

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if an edge with length equal to *exactLength* has an adjacent edge with length equal to *exactAdjLength*.

'exceptRectangle

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply on rectangular shapes.

'fillerOnly

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the minimum edge length rule is considered only in filler insertions in placement.

f_edgeLength

The constraint does not apply if the sum of the consecutive edges that are shorter than *edgeLength* is less than this value.

Examples

- Example 1: minStepEdgeLength with adjEolSpacingRange
- Example 2: minStepEdgeLength with horizontalEdge, convexCorner, and mixedCorner
- Example 3: minStepEdgeLength with protrusionEdge
- Example 4: minStepEdgeLength with jogEdge
- Example 5: minStepEdgeLength with adjJogEdge
- Example 6: minStepEdgeLength with adjNotchEdge
- Example 7: minStepEdgeLength with extendToAlign
- Example 8: minStepEdgeLength with exceptExactLength and exceptExactBothLength

Virtuoso Technology Data Constraints Reference

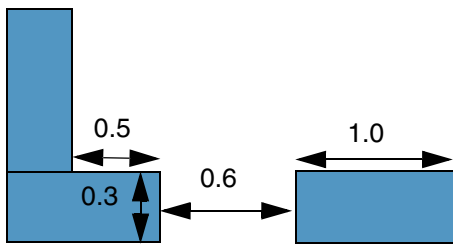
Length and Width Constraints

Example 1: *minStepEdgeLength* with *adjEolSpacingRange*

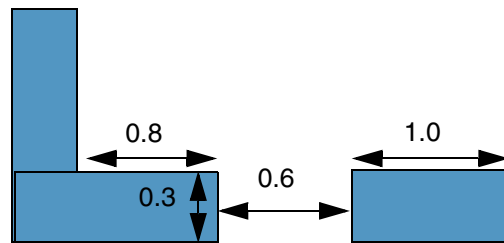
The constraint applies to an edge if any one of its adjacent edges is an end-of-line edge with width equal to 0.3 and this adjacent edge has another perfectly aligned end-to-end-shape within the *adjEolSpacingRange* value of 0.6.

```
spacings (
  ( minStepEdgeLength "Metal1"
    'width 0.3
    'adjEolSpacingRange 0.6
    0.7
  )
) ;spacings
```

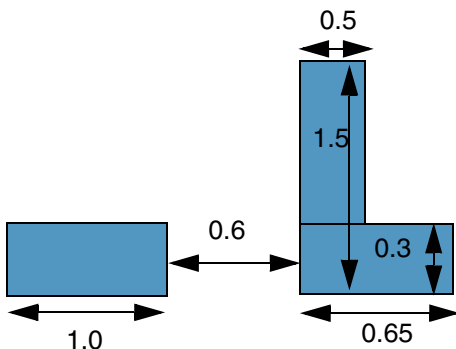
■ Metal1



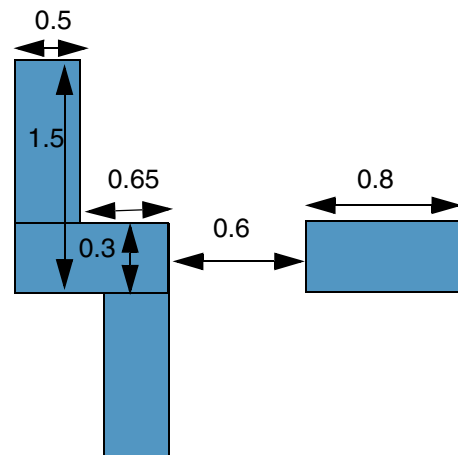
The constraint fails because the edge (0.5) meets the *adjEolSpacingRange* requirements and its length is 0.5, which is less than 0.7.



The constraint passes because the edge (0.8) meets the *adjEolSpacingRange* requirements and its length is 0.8, which is more than 0.7.



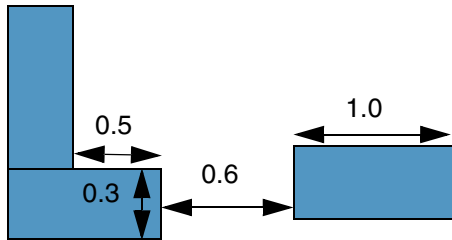
The constraint does not apply because the joint is not an end-of-line.



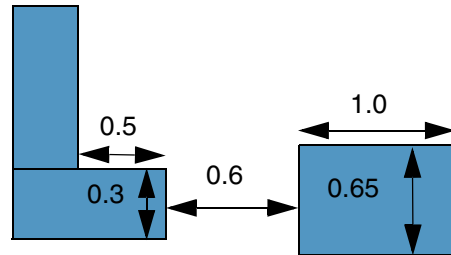
The constraint does not apply because the joint is not an end-of-line.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints



The constraint does not apply because the other end-to-end shape is not perfectly aligned.



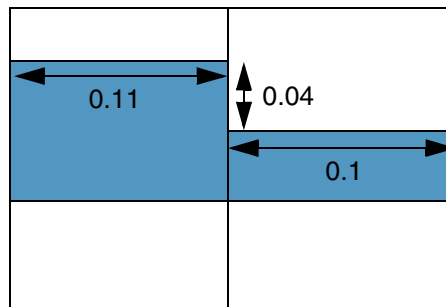
The constraint does not apply. The centers of the two line-ends are aligned but, due to different widths, the two ends are not perfectly aligned.

Example 2: *minStepEdgeLength* with *horizontalEdge*, *convexCorner*, and *mixedCorner*

On layer Metal1, horizontal edges between convex corners must be at least 0.12 long and horizontal edges between a concave and a convex corner must be at least 0.1 long.

```
spacings (
  ( minStepEdgeLength "Metal1"
    'horizontalEdge 'convexCorner
    0.12
  )
  ( minStepEdgeLength "Metal1"
    'horizontalEdge 'mixedCorner
    0.1
  )
) ;spacings
```

■ Metal1



FAIL. The right horizontal edge lies between a concave and a convex corner and is 0.1 long, but the length of the left horizontal edge, which lies between two convex corners, is less than the required minimum length of 0.12.

Virtuoso Technology Data Constraints Reference

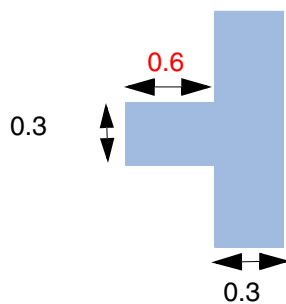
Length and Width Constraints

Example 3: *minStepEdgeLength* with *protrusionEdge*

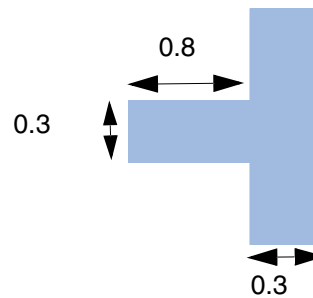
The constraint applies only to a protrusion edge.

```
spacings (
  ( minStepEdgeLength "Metal1"
    'width 0.3
    'mixedCorner
    'horizontalEdge
    'protrusionEdge
    0.7
  )
) ;spacings
```

■ Metal1



FAIL. The constraint applies to the horizontal protrusion edge. The constraint fails because the edge length is 0.6 which is less than 0.7.



PASS. The constraint applies to the horizontal protrusion edge. The constraint passes because the edge length is 0.8 which is greater than 0.7.

Virtuoso Technology Data Constraints Reference

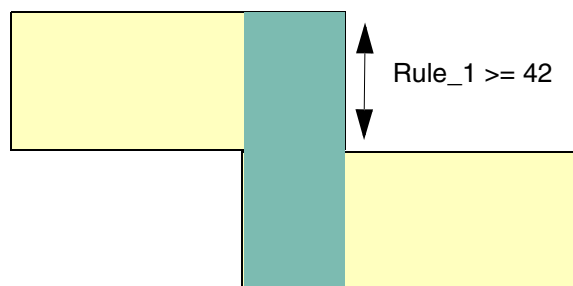
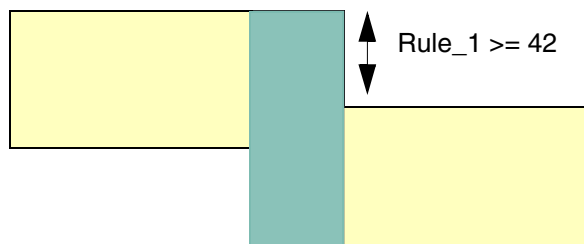
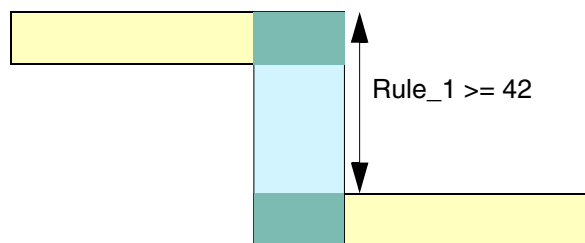
Length and Width Constraints

Example 4: *minStepEdgeLength* with *jogEdge*

The constraint applies only to a jog edge or a Z shape.

```
spacings(  
  ( minStepEdgeLength "Metal1"  
    'verticalEdge  
    'mixedCorner  
    'jogEdge 0.042  
    'ref "Rule_1"  
  )  
); spacings
```

 Vertical wire
 Horizontal wire

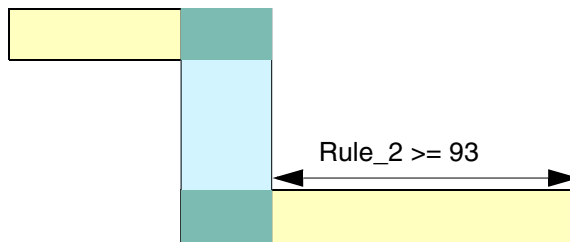


Example 5: *minStepEdgeLength* with *adjJogEdge*

The constraint applies only to an edge adjacent to a jog edge.

```
spacings (
  ( minStepEdgeLength "Metal1"
    'verticalEdge
    'mixedCorner
    'adjJogEdge 0.093
    'ref "Rule_2"
  )
) ;spacings
```

Vertical wire
 Horizontal wire

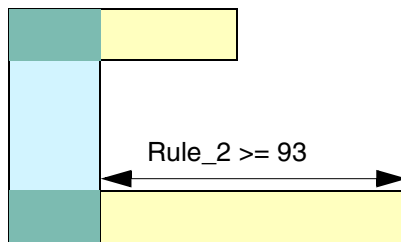


Example 6: *minStepEdgeLength* with *adjNotchEdge*

The constraint applies only to an edge adjacent to a notch edge.

```
spacings (
  ( minStepEdgeLength "Metal1"
    'verticalEdge
    'mixedCorner
    'adjNotchEdge 0.093
    'ref "Rule_2"
  )
) ;spacings
```

Vertical wire
 Horizontal wire



Example 7: *minStepEdgeLength* with *extendToAlign*

On layer Implant1, horizontal edges between two convex corners must be at least 0.04. Shapes with width 0.05 with a gap less than or equal to 0.07 will be merged.

```
spacings (
  (minStepEdgeLength "Implant1"
```

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

```
'convexCorner  
'horizontalEdge  
'extendToAlign (0.05 0.07)  
0.04)  
) ;spacings
```

Virtuoso Technology Data Constraints Reference

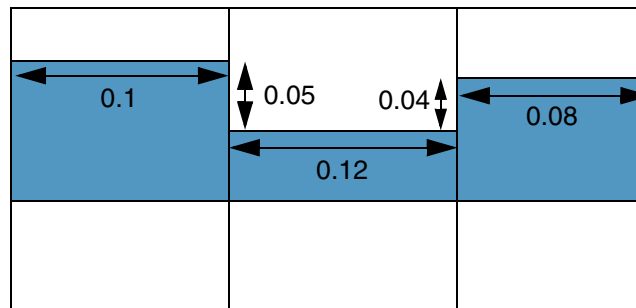
Length and Width Constraints

Example 8: *minStepEdgeLength* with *exceptExactLength* and *exceptExactBothLength*

All horizontal edges on layer Metal1 must be at least 0.12 long. The constraint does not apply if a 0.1 long horizontal edge has an adjacent edge with length less than or equal to 0.6 or if a 0.08 long horizontal edge has an adjacent edge with length equal to 0.04.

```
spacings (
  ( minStepEdgeLength "Metal1"
    'horizontalEdge
    'exceptExactLength ((0.1 0.06))
    'exceptExactBothLength ((0.08 0.04))
    0.12
  )
) ;spacings
```

■ Metal1



PASS. The left horizontal edge has an adjacent edge with length 0.05 (<0.06), and is, therefore, exempt; the middle horizontal edge is 0.12 long, and, therefore, satisfies the constraint; the right horizontal edge has an adjacent edge that is equal to 0.04, and is, therefore, also exempt.

minWidth

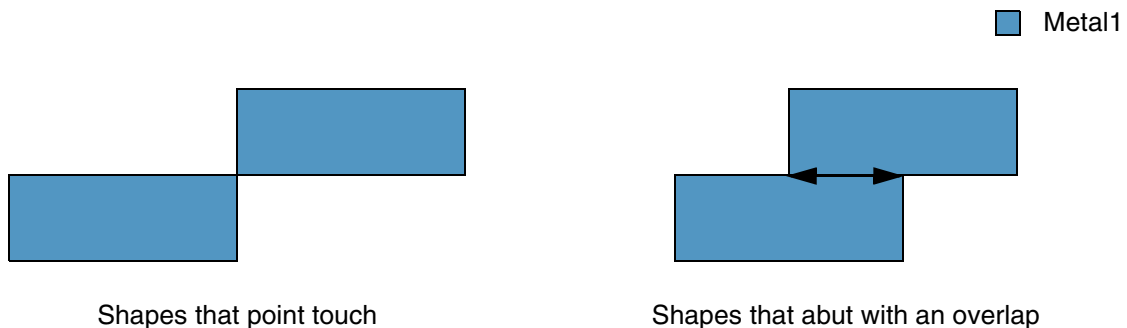
```

spacings (
  ( minWidth tx_layer
    ['exceptPointTouch 'exceptLineTouch (g_range)]
    ['vertical | 'horizontal]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['length f_length]
    f_width ['manhattan']
  )
) ;spacings

```

Specifies the minimum orthogonal width for any shape on the specified layer.

A shape can point touch a second shape on the same layer at a corner. A shape can also abut a second shape on the same layer with an overlap if the overlap value falls in the exempt ranges.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_width</i>	The orthogonal width of the shape must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

`'exceptPointTouch` The shape can point touch a second shape on the same layer at a corner.

Note: Both shapes must satisfy the constraint.

Type: Boolean

`'exceptLineTouch (g_range)`

The shape can abut a second shape with an overlap that falls in this range.

Note: Both shapes must satisfy the constraint.

Type: Floating-point values specifying a range of overlap values that are exempt

`'horizontal | 'vertical`

The direction in which width is measured. If direction is not specified, width is measured in any direction.

Note: This parameter can be used to model the width of a shape in a direction perpendicular to the routing direction specified for the layer.

Type: Boolean

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'length f_length` (Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to shapes with length less than this value.

`'manhattan` The constraint uses Manhattan distance, which allows a larger spacing at the corners.

By default, the constraint uses Euclidean measurement.

Type: Boolean

Note: The `'horizontal` and `'vertical` parameters are supported by the Generating All Components from Source and Update Components And Nets commands.

Virtuoso Technology Data Constraints Reference

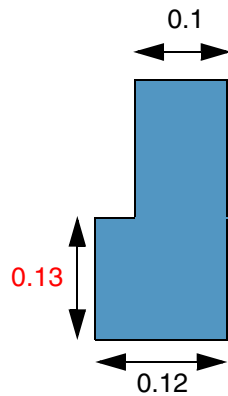
Length and Width Constraints

Example

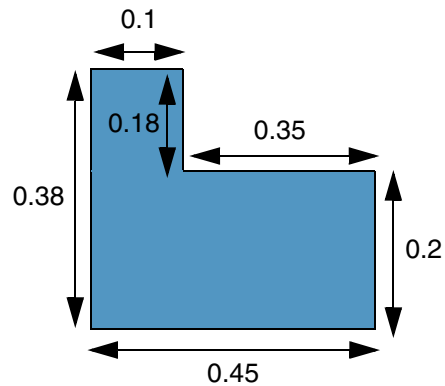
The width of a shape when measured horizontally must be at least 0.1 and when measured vertically must be at least 0.2.

```
spacings(  
  ( minWidth "Metal1"  
    'horizontal  
    0.1  
  )  
  ( minWidth "Metal1"  
    'vertical  
    0.2  
  )  
) ;spacings
```

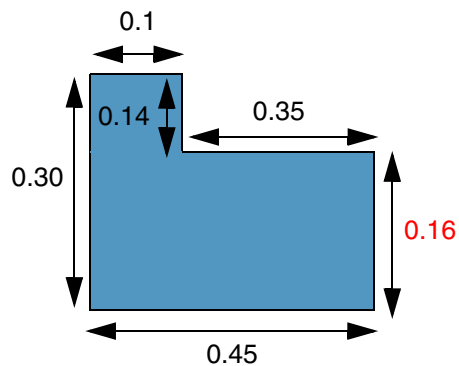
■ Metal1



a) FAIL. The 0.13 width measured vertically is less than the required minimum width of 0.2 in that direction.



b) PASS. The 0.18 width does not count as a vertical width.

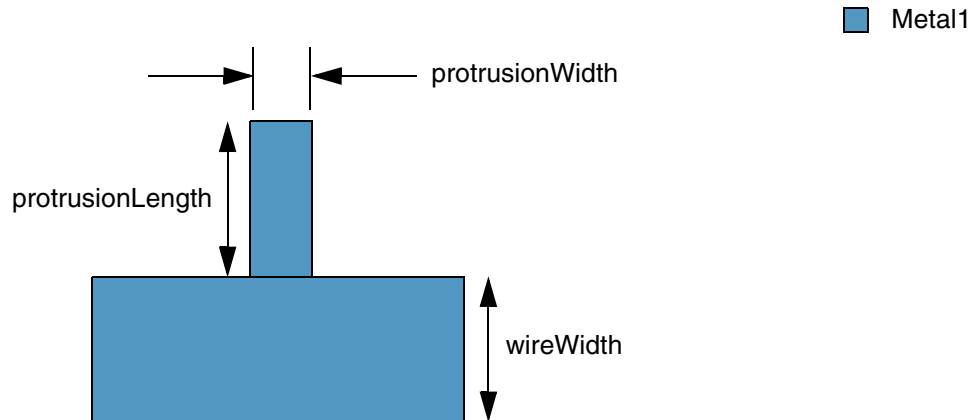


c) FAIL. The 0.16 width measured vertically is less than the required minimum width of 0.2 in that direction.

protrusionWidth

```
spacings (
    ( protrusionWidth tx_layer
      ( f_protrusionLength f_wireWidth f_protrusionWidth )
    )
) ;spacings
```

Specifies the minimum width for a protrusion that connects to a wire. The width of the protrusion depends on the length of the protrusion and the width of the wire to which it connects.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_protrusionLength</i>	The constraint applies only if the length of the protrusion is less than this value.
<i>f_wireWidth</i>	The constraint applies only if the width of the wire to which the protrusion connects is greater than or equal to this value.
<i>f_protrusionWidth</i>	The width of the protrusion must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Length and Width Constraints

Parameters

None

Example

On Metal3, the width of a protrusion must be greater than or equal to 0.25 if the length of the protrusion is less than 1.5 and the width of the wire to which this protrusion connects is greater than or equal to 0.5.

```
spacings(  
    ( protrusionWidth "Metal3" (1.5 0.5 0.25) )  
) ;spacings
```

Miscellaneous Constraints

This topic includes the following constraints:

- [allowedShapeAngles](#)
- [allowedSnapPatternDefs](#)
- [allowedWireTypes](#)
- [connectLayers](#)
- [coreEOLBlockage](#)
- [diagonalShapesAllowed](#)
- [edgeMustCoincide](#)
- [edgeMustOverlap](#)
- [errorLayer](#)
- [fixedMask](#)
- [gateOrientation](#)
- [ignoreShieldingOnLayers](#)
- [implantGroup](#)
- [integrationColorModel](#)
- [layerDefaultColor](#)
- [layerProp](#)
- [layerThickness](#)
- [lefLayerOrder](#)
- [lefType](#)
- [libProp](#)

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

- maxACCurrentDensity
- maxNumCorners
- minOneDArrayStructure
- minStitchOverlap
- msCrossTieLayers
- msLayerCrossTieInterval
- msSameMask
- msShieldingLimit
- msShieldStyle
- multiMaskCheck
- overlapViaGroup
- pinConnectBlockage
- preColoredLayers
- rectShapeDir
- sameMaskOnLayer
- trimMetalTrack
- trimShape
- useExistingShapesAsShield
- validCutClass
- vertexInsideForbidden
- voltageMode
- widthLengthRatio

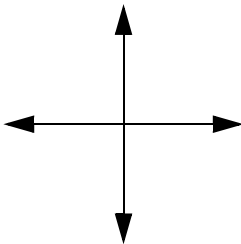
allowedShapeAngles

```
allowedShapeAngles (
  ["anyAngle" | "orthogonal" | "diagonal"]
  (tx_layer {"anyAngle" | "orthogonal" | "diagonal"})
  ...
) ;allowedShapeAngles
```

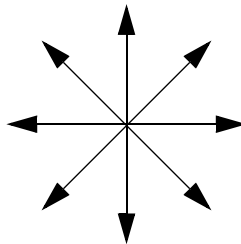
Sets the angle or angles at which a shape on a specified layer can be placed. It also allows specification of the default angles for layers not specifically assigned angles.

The arrows illustrate the angles allowed by each setting.

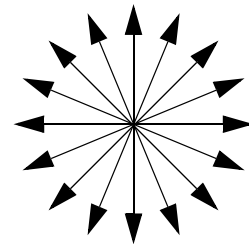
`orthogonal` allows vertical and horizontal shapes.



`diagonal` allows vertical and horizontal shapes and shapes at 45° and 135°.



`anyAngle` allows shapes at any angle, including but not restricted to these.



Values

"anyAngle" | "orthogonal" | "diagonal"

The angles at which a shape can be placed:

- `anyAngle`: Shapes can be placed at any angle.
- `orthogonal`: Shapes must be placed orthogonally.
- `diagonal`: Shapes can be placed orthogonally or diagonally at 45 or 135 degrees.

`tx_layer`

The layer on which the angle setting is applied.

Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Parameters

None

Example

The allowed shape angle for Metal1 is set to `anyAngle`, for Metal2 to `diagonal`, and for all other layers to `orthogonal`.

```
allowedShapeAngles (
    "orthogonal"
    ( "Metal1" "anyAngle")
    ( "Metal2" "diagonal")
) ;allowedShapeAngles)
```

allowedSnapPatternDefs

```
spacings (
    ( allowedSnapPatternDefs
      (t_snapPatternDef ...)
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the snapPatternDefs and widthSpacingSnapPatternDefs allowed in a cellview.

Values

t_snapPatternDef

Allowed snap pattern definitions. Every entry references a snapPatternDef or widthSpacingSnapPatternDef by name.

Type: A list of snap pattern definitions, each enclosed in double quotes and separated by a space

Parameters

None

Example

The only snapPatternDefs allowed in a cellview are M2_globalGrid and M3_globalGrid.

```
spacings (
    ( allowedSnapPatternDefs
      ("M2_globalGrid" "M3_globalGrid")
    )
) ;spacings
```

allowedWireTypes

```
spacings(  
    ( allowedWireTypes tx_layer  
      (t_wireType ...)  
    )  
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) 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).

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t_wireType</i>	Allowed wire types. Type: A list of wire types, each enclosed in double quotes and separated by a space

Parameters

None

Example

Wire types 1X, 2X, and 3X can be set for tracks on layer Metal3.

```
spacings(  
    ( allowedWireTypes "Metal3"  
      ("1X" "2X" "3X")  
    )  
) ;spacings
```


connectLayers

```
spacings(  
    (connectLayers (tx_layer1 tx_layer2 ... tx_layerN)  
        ['connectBy tx_connectBy_layer']  
    )  
); spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the layers that are considered to be electrically connected by the Virtuoso connectivity extractor.

Values

tx_layer1 tx_layer2 ... tx_layerN

The layers on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

Parameters

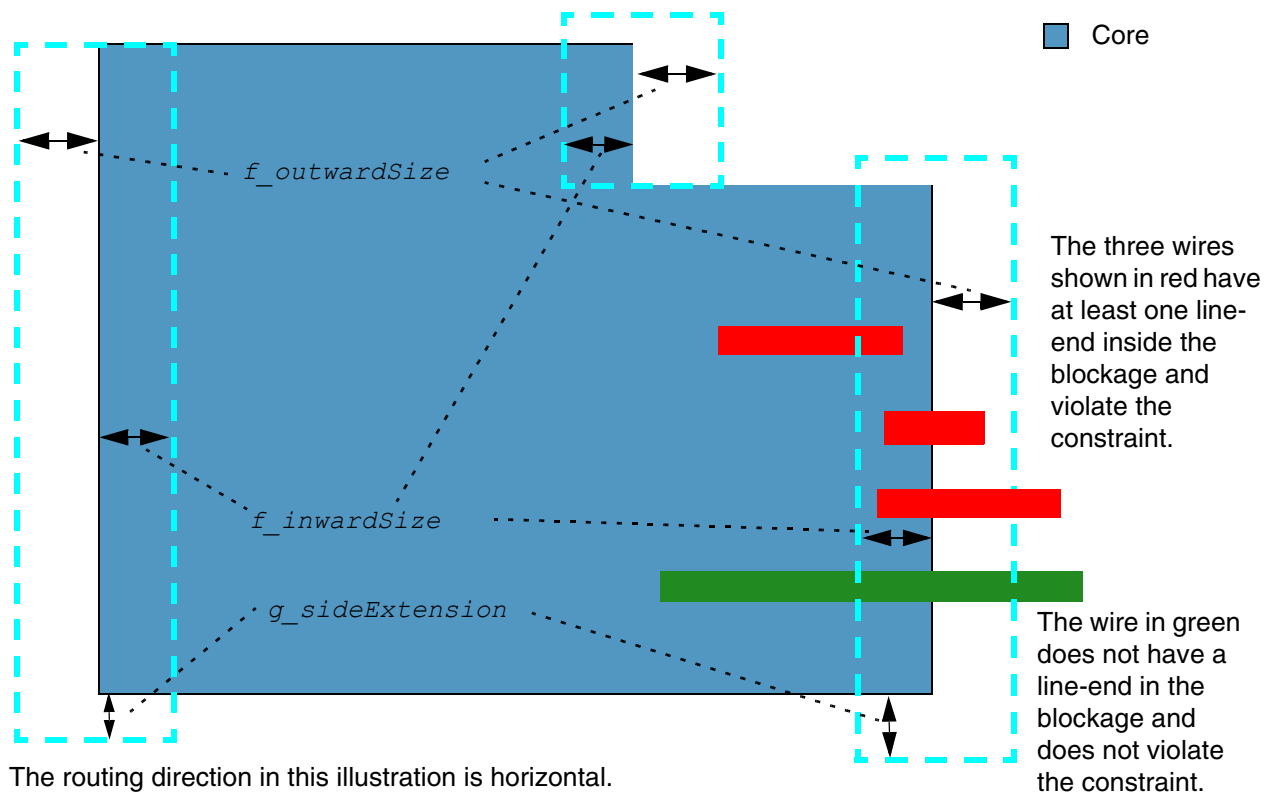
'connectBy tx_connectBy_layer

The layer with which the connectivity is established.

coreEOLBlockage

```
spacings (
  ( coreEOLBlockage tx_layer
    {'horizontal' | 'vertical'}
    ['sideExtension g_sideExtension']
    (f_outwardSize f_inwardSize)
  )
) ; spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the size and direction of end-of-line blockages from core edges. Wires starting in the core interior and ending with an end-of-line edge in the blockage violate the constraint. On the other hand, wires with no end-of-line edge in the blockage or those that start in the edge but end in the blockage do not violate the constraint.



Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_outwardSize f_inwardSize</i>	The <i>f_outwardSize</i> value specifies the size of the line-end blockage measured outward from the core. The <i>f_inwardSize</i> value specifies the size of the line-end blockage measured inward from the core. Type: Float, specified in microns.

Parameters

<i>'horizontal 'vertical</i>	End-of-line blockages are formed on core edges in this direction.
<i>'sideExtension g_sideExtension</i>	(Virtuoso Layout Suite EXL and higher tiers) Line-end blockages are extended by the given <i>g_sideExtension</i> value along the direction of the core edge.

Example

Specifies that the blockage area is orthogonal to the vertical direction by 0.06 microns outward from the core edge and 0.08 microns inward from the core edge. Line-end blockages are extended by 0.04 along the direction of the core edge.

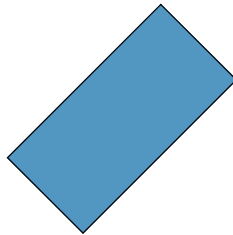
```
spacings(  
    (coreEOLBlockage "Metall1"  
        'vertical  
        'sideExtension 0.04  
        (0.06 0.08))  
) ;spacings
```

diagonalShapesAllowed

```
spacings(  
    ( diagonalShapesAllowed tx_layer { t | nil } )  
) ;spacings
```

Specifies whether shapes at a 45 degree angle are allowed on the specified layer.

 Metal1



If set to `t`, shapes can be placed diagonally at 45 degrees.

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>t nil</code>	If <code>t</code> , shapes can be placed diagonally at 45 degrees. If <code>nil</code> , shapes cannot be placed diagonally.

Parameters

None

Example

The following constraint definition enables 45 degree diagonal shapes on layer poly2.

```
spacings(  
    ( diagonalShapesAllowed "poly2" t )  
) ;spacings
```

edgeMustCoincide

```
orderedSpacings (
    ( edgeMustCoincide tx_layer1 tx_layer2
      ['horizontalEdge | 'verticalEdge]
      ['edgeLengthRanges (g_ranges) ['endOfLineOnly]]
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the *layer1* edges that must coincide with a *layer2* edge.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

'horizontalEdge 'verticalEdge	The constraint applies only to <i>layer1</i> edges that are in this direction. Type: Boolean
'edgeLengthRanges (<i>g_ranges</i>)	The constraint applies only if the <i>layer1</i> edge length is in this range. Type: Floating-point values specifying the length <u>ranges</u> to which the constraint applies
'endOfLineOnly	The constraint applies only if the <i>layer1</i> edge is between two convex corners and its length falls in the range specified with 'edgeLengthRanges. Type: Boolean

Virtuoso Technology Data Constraints Reference

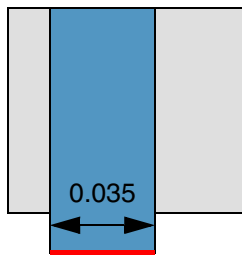
Miscellaneous Constraints

Example

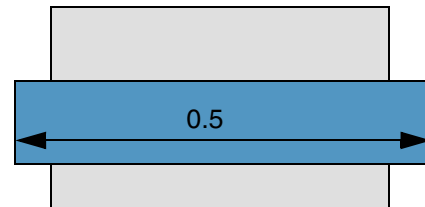
Horizontal Metal1 edges between two convex corners must coincide with a Metal2 edge if the edge length is less than 0.02 or greater than 0.03 and less than 0.04.

```
orderedSpacings (
  ( edgeMustCoincide "Metal1" "Metal2"
    'horizontalEdge
    'edgeLengthRanges ("<0.02" "(0.03 0.04)")
    'endOfLineOnly
  )
) ;orderedSpacings
```

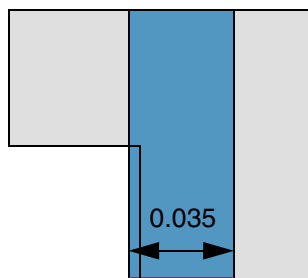
□ Metal2
■ Metal1



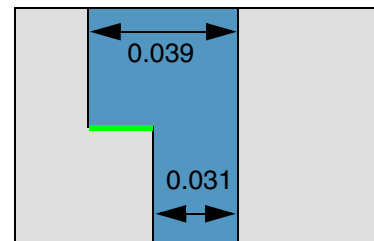
a) FAIL. The Metal1 horizontal edges are 0.035 long, a value that falls in the specified range. Therefore, the constraint applies, but is not met because the bottom edge is not coincident with a Metal2 edge.



b) The constraint does not apply because the length of the Metal1 horizontal edges is 0.5, a value that does not fall in the specified range.



c) FAIL. The Metal1 horizontal edges are 0.035 long, a value that falls in the specified range. Therefore, the constraint applies, but is not met because the bottom edge is not fully covered by the Metal2 edge.



d) PASS. The edge in green is not an end-of-line edge (that is, it is not between two convex edges). The other two horizontal Metal1 edges have lengths in the specified range and coincide with Metal2 edges.

edgeMustOverlap

```
orderedSpacings (
    ( edgeMustOverlap tx_layer1 tx_layer2
      ['horizontalEdge | 'verticalEdge]
      ['edgeLengthRanges (g_ranges) ['endOfLineOnly]]
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies that edges on *layer1* must be overlapped by a shape on *layer2*.

This constraint requires an overlap, but does not specify the extent of the required overlap, which can be specified with the `minDirectionalOverlap` constraint.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

'horizontalEdge 'verticalEdge	The constraint applies only to <i>layer1</i> edges that are in this direction. Type: Boolean
'edgeLengthRanges (<i>g_ranges</i>)	The constraint applies only if the <i>layer1</i> edge length is in this range. Type: Floating-point values specifying the length <u>ranges</u> to which the constraint applies

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

'endOfLineOnly	The constraint applies only if the <i>layer1</i> edge is between two convex corners and its length falls in the range specified with 'edgeLengthRanges. Type: Boolean

Examples

- Example 1: edgeMustOverlap with horizontalEdge and edgeLengthRanges
- Example 2: edgeMustOverlap with horizontalEdge, endOfLineOnly, and edgeLengthRanges

Virtuoso Technology Data Constraints Reference

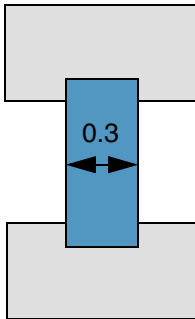
Miscellaneous Constraints

Example 1: *edgeMustOverlap* with *horizontalEdge* and *edgeLengthRanges*

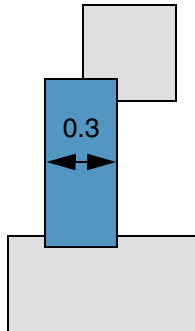
All horizontal Metal1 edges that are less than or equal to 0.5 long must be fully overlapped by a Metal2 shape.

```
orderedSpacings (
  ( edgeMustOverlap "Metal1" "Metal2"
    'horizontalEdge
    'edgeLengthRanges ("<=0.5")
  )
) ;orderedSpacings
```

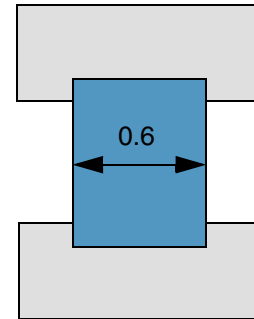
□ Metal2
■ Metal1



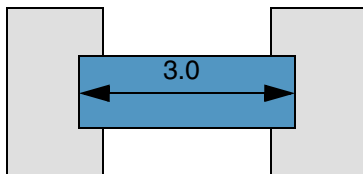
a) PASS. The constraint applies to the Metal1 horizontal edges (length ≤ 0.5) and both these edges are fully overlapped by Metal2 shapes.



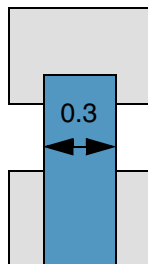
b) FAIL. The constraint applies, but the top horizontal Metal1 edge is not fully overlapped by the Metal2 shape.



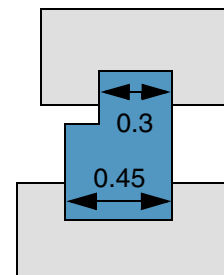
c) The constraint does not apply because the length of the Metal1 horizontal edges is 0.6 (> 0.5).



d) The constraint does not apply because the length of the Metal1 horizontal edges is 3.0 (> 0.5).



e) FAIL. The constraint applies, but is not met because zero overlap is not allowed (the bottom Metal1 edge is not fully overlapped by the Metal2 shape).



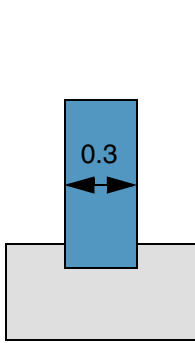
f) FAIL. The constraint applies, but is not met because the middle horizontal Metal1 edge is not fully overlapped by a Metal2 shape.

Example 2: *edgeMustOverlap* with *horizontalEdge*, *endOfLineOnly*, and *edgeLengthRanges*

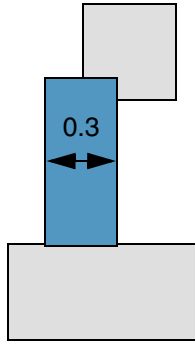
All horizontal Metal1 edges that are between two convex corners (end-of-line edges) and are less than or equal to 0.5 long must be fully overlapped by a Metal2 shape.

```
orderedSpacings (
  (edgeMustOverlap "Metal1" "Metal2"
    'horizontalEdge
    'edgeLengthRanges ("<=0.5")
    'endOfLineOnly
  )
) ;orderedSpacings
```

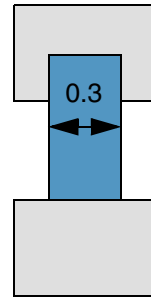
□ Metal2
■ Metal1



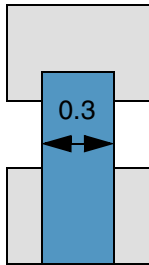
a) FAIL. The top Metal1 edge is not overlapped by a Metal2 shape.



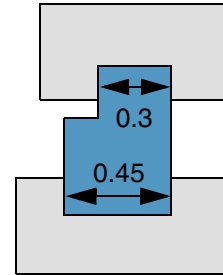
b) FAIL. The top Metal1 edge is not fully overlapped by the Metal2 shape.



c) FAIL. The bottom Metal1 edge is not overlapped by the Metal2 shape (zero overlap).



d) FAIL. The bottom Metal1 edge is not overlapped by the Metal2 shape (zero overlap).



e) PASS. The two end-of-line Metal1 horizontal edges are fully overlapped by Metal2 shapes.

errorLayer

```
interconnect (
    ( errorLayer (tx_derivedLayer)
      ...
    )
) ;interconnect
```

Specifies whether it is an error for geometries to exist on the specified derived layer.

Values

tx_derivedLayer The derived layer on which the constraint is applied.

Parameters

None

Example

It is not legal for shapes to exist on derived layers named derivedLayer1 and derivedLayer2.

```
interconnect (
    ( errorLayer "derivedLayer1" )
    ( errorLayer "derivedLayer2" )
) ;interconnect
```

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

fixedMask

```
spacings (
    ( fixedMask )
) ; spacings
```

Prevents mask shifting for cellviews defined in the library.

Values

None

Parameters

None

gateOrientation

```
spacings(  
    ( gateOrientation tx_derivedLayer  
      ['width f_width]  
      {"horizontal" | "vertical" | "any"}  
    )  
); spacings
```

Specifies the orientation for gates on the specified derived layer.

For some processes, all gates with width less than or equal to the specified value must be created in the same orientation, horizontal or vertical.

Values

<i>tx_derivedLayer</i>	The derived layer on which the constraint is applied.
"horizontal" "vertical" "any"	The orientation of the gate. 'any' permits either orientation, horizontal or vertical.

Parameters

<i>f_width</i>	The constraint applies only to the gates that have width less than or equal to this value. Otherwise, the constraint applies to gates of any width.

Virtuoso Technology Data Constraints Reference

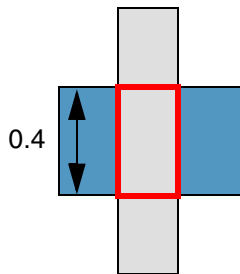
Miscellaneous Constraints

Example

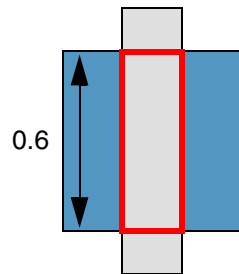
Orientation must be horizontal for gates with width less than or equal to 0.5.

```
spacings(  
  ( gateOrientation "Gate1"  
    'width 0.5  
    "horizontal"  
  )  
) ;spacings
```

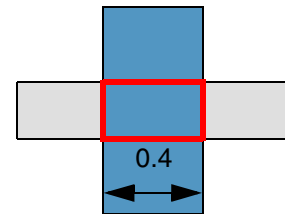
□ Poly
■ Diffusion
□ Gate1: Derived layer for gate shapes



a) FAIL. The width of the gate is 0.4 (<0.5), but its orientation is vertical.



b) The constraint does not apply because the width of the gate is 0.6 (>0.5).



c) PASS. The width of the gate is 0.4 (<0.5) and its orientation is horizontal.

ignoreShieldingOnLayers

```
spacings(  
    ( ignoreShieldingOnLayers (tx_layer1 [tx_layer2 ... tx_layerN]) )  
) ;spacings
```

Requires that no shielding be added on the specified layers.

Values

tx_layer1 tx_layer2 ... tx_layerN

The layers on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

Parameters

None

Example

Shielding should not be added on layers Metal1 and Metal2.

```
spacings(  
    ( ignoreShieldingOnLayers ("Metal1" "Metal2") )  
) ;spacings
```

implantGroup

```
spacings(  
    ( implantGroup tx_layer1 [tx_layer2 ... tx_layerN]  
      'basedLayer tx_layer  
        t_groupName  
      )  
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies an implant group. Shapes on the specified layers should be combined and then the rules of the specified `basedLayer` should be checked on the group.

Values

tx_layer1 [*tx_layer2* ... *tx_layerN*]

The layers on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

t_groupName

The name of the implant group.

Parameters

'basedLayer *tx_layer*

The combined implant shapes need to meet the constraints on this layer.

Example

Shapes on the specified layers are combined and then the rules of the specified `basedLayer` are checked on the group.

```
spacings(  
    ( implantGroup VT1 VT2 'basedLayer VT2 "GROUP1" )  
    ( implantGroup VT1 VT3 VT4 'basedLayer VT1 "GROUP2" )  
); spacings
```


integrationColorModel

```
colorControl(  
    ( integrationColorModel "locked" | "any"  
    )  
) ;colorControl
```

Specifies that instances of blocks bound to a technology or instances of a specific block can have color shifting applied to them.



`colorControl` is similar to other constraint categories such as `spacings`, `orderedSpacings`, and `routingGrids`.

Values

"locked" | "any"

The value `any` signifies that any kind of color shifting can be applied. This is the default value that also applies if the constraint is not specified.

The value `locked` signifies that coloring applications should not apply color shifting. Type: String (layer and purpose names) or Integer (layer number)

Parameters

None

layerDefaultColor

```
spacings(  
    ( layerDefaultColor tx_layerName tx_color  
    )  
); spacings
```

(Virtuoso Advanced Node for Layout Standard) Sets the default mask color for all purposes of a layer.

Values

<i>tx_layerName</i>	Name of the layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_color</i>	Default mask color for the layer. Valid values: gray, mask1, mask2, mask3, and mask4

Parameters

None

Example

The default mask color for layer Metall is mask1.

```
spacings(  
    ( layerDefaultColor "Metall"  
      "mask1"  
    )  
); spacings
```

layerProp

```
spacings(  
    ( layerProp tx_layer  
        'propName t_propName  
        t_propValue  
    )  
); spacings
```

Specifies a property and value for a LEF layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t_propValue</i>	The value of the specified property for the specified layer. Type: String.

Parameters

<i>'propName t_propName</i>	The name of the LEF property. Type: String.

Example

The LEF58_SPACING property for layer Metall1 includes as its value the syntax for the SAMEMASK spacing.

```
spacings(  
    ( layerProp "Metall1"  
        'propName "LEF58_SPACING"  
        "SPACING 0.2 SAMEMASK ;"  
    )  
); spacings
```

layerThickness

```
spacings(  
    ( layerThickness tx_layer  
      g_thicknessValue  
    )  
) ;spacings
```

Specifies the thickness of a layer.

Values

<i>tx_layer</i>	The layer to which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t_thicknessValue</i>	The thickness of the specified layer. Type: Float.

Example

The constraint sets a layer thickness of 0.04 to Metal1.

```
spacings(  
    ( layerThickness "Metal1"  
      0.04  
    )  
) ;spacings
```

lefLayerOrder

```
spacings(  
    ( lefLayerOrder l_layerNames  
    )  
); spacings
```

Specifies the order of layers in the original LEF.

Values

<i>l_layerNames</i>	Layer names in the order in which they appear in the original LEF.
	Type: String.

lefType

```
spacings(  
    ( lefType t_layer  
      ['lef58Type t_lef58Type]  
      t_lefType  
    )  
); spacings
```

Specifies the material type of the LEF layer from the base PDK.

Values

<i>t_layer</i>	The name of the LEF layer to which the constraint is applied. Type: String.
<i>t_lefType</i>	The material type of the LEF layer in the base PDK. Type: String.

Parameters

<i>'lef58Type t_lef58Type</i>	The value of the LEF58_TYPE property in the LEF. Type: String.

Example

The base material MASTERSLICE is specified for layer TM1. Additionally, the LEF58_TYPE property value is specified as TYPE TRIMMETAL ;.

```
spacings(  
    ( lefType "TM1"  
      'lef58Type "TYPE TRIMMETAL ;"  
      "MASTERSLICE"  
    )  
); spacings
```

libProp

```
spacings(  
    ( libProp  
        'propName t_propName  
        ['int | 'float] t_propValue  
    )  
); spacings
```

Specifies a value for a LEF library property.

Values

<i>t_propValue</i>	The value of the specified LEF library property. Type: String (default) or <code>int</code> or <code>float</code> , if specified.

Parameters

<code>'propName</code> <i>t_propName</i>	The name of the LEF library property. Type: String
<code>'int</code> <code>'float</code>	If <code>int</code> or <code>float</code> is not specified, the value is assumed to be string.

Example

The values for the `LEF58_TRIMMETALTRACK` and `versionNum` library properties are specified.

```
spacings(  
    ( libProp  
        'propName "LEF58_TRIMMETALTRACK"  
        "TRIMMETALTRACK M1TRIM COREOFFSET 0.1 PITCH 0.2"  
    )  
    ( libProp  
        'propName "versionNum" 'int "12"  
    )  
); spacings
```

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

maxACCurrentDensity

```
layerRules(  
  currentDensity(  
    ( rmsACCurrentDensity tx_layer f_value )  
  ) ;currentDensity  
  
  currentDensityTables(  
    ( rmsACCurrentDensity tx_metalLayer  
      (( "frequency" nil nil ["width" nil nil] ))  
      ( g_table )  
    )  
    ( rmsACCurrentDensity tx_cutLayer  
      (( "frequency" nil nil ["cutArea" nil nil] ))  
      ( g_table )  
    )  
  ) ;currentDensityTables  
) ;layerRules  
  
spacings(  
  ( maxACCurrentDensity tx_layer  
    {'peak | 'rms | 'average}  
    ['tempFactor ( (( "temperature" )) (f_temp f_mult ...) )]  
    ['hoursFactor ( (( "hours" )) (f_hour f_mult ...) )]  
    f_value  
  )  
) ;spacings  
  
spacingTables(  
  ( maxACCurrentDensity tx_metalLayer  
    (( "frequency" nil nil ["width" nil nil] )  
    {'peak | 'rms | 'average}  
    ['tempFactor ( (( "temperature" )) (f_temp f_mult ...) )]  
    ['hoursFactor ( (( "hours" )) (f_hour f_mult ...) )]  
    [f_default]  
  )  
  (g_table)  
) ;maxACCurrentDensity  
  
  ( maxACCurrentDensity tx_cutLayer  
    (( "frequency" nil nil ["cutArea" nil nil] )  
    {'peak | 'rms | 'average}  
    ['tempFactor ( (( "temperature" )) (f_temp f_mult ...) )]  
    ['hoursFactor ( (( "hours" )) (f_hour f_mult ...) )]  
    [f_default]  
  )  
  (g_table)  
) ;maxACCurrentDensity  
) ;spacingTables
```


Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Defines the maximum allowable AC current density.

In some processes, the allowable AC current density is either defined as a simple value or is dependent on the frequency and width of the shape through which the current is flowing. In more advanced processes, the allowable current density can be a function of the operating temperature and the number of hours for which the circuit is in operation. The temperature is measured in degree Celsius.

When the operating temperature is greater than or equal to a given temperature, the corresponding multiplier is applied to the current value obtained from the constraint. The length of time of circuit operation is measured in number of hours. When the number of operating hours is greater than or equal to a certain value, the corresponding multiplier is applied to the current value obtained from the constraint (for a given frequency and width if the current density is defined as a function of these parameters).

Values

tx_layer, tx_metalLayer, tx_cutLayer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_value

The maximum allowable AC current density.

"frequency" nil nil ["width" nil nil]

This identifies the index for *table*.

"frequency" nil nil ["cutArea" nil nil]

This identifies the index for *table*.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

g_table

Each row in the current density table has the following format:

```
( f_frequency f_value f_currentValue )
```

where,

- *f_frequency* is the frequency, in megahertz. The value must be greater than or equal to the index value for the corresponding current density value to apply.
- *f_value* is width if the layer specified is a metal layer and area if the layer specified is a cut layer. The value must be greater than or equal to the index value for the corresponding current density value to apply.
- *f_currentValue* is the maximum allowable AC current density, in milliamperes per micron.

Type: A 1-D table specifying floating-point frequency and current density values, or a 2-D table specifying floating-point frequency and width, or area, and current density values

Parameters

'peak | 'rms | 'average

The current density type.

- 'peak: The peak AC current density that the wire can handle.
- 'rms: The RMS AC current density that the wire can handle.
- 'average: The average AC current density that the wire can handle.

Type: Boolean

```
'tempFactor ( ( ( "temperature" ) ) ( f_temp f_mult ... ) )
```

A multiplier table that is indexed on the temperature. When the temperature is greater than or equal to an index value in the table, the current is scaled by multiplying it with the corresponding multiplier.

Type: A 1-D table specifying temperature in degree Celsius and a unitless multiplicative factor

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

`'hoursFactor ( ( ( "hours" ) ) ( f_hour f_mult ... ) )`

A multiplier table that is indexed on the number of hours of operation. When the number hours of operation is greater than or equal to an index value in the table, the current is scaled by multiplying it with the corresponding multiplier.

Type: A 1-D table specifying the number of hours and a unitless multiplicative factor

`f_default`

The AC current density value to be used when no table entry applies.

Example

Define the peak current density by using a piecewise model at 100 Mhz and 400 MHz for widths 0.3, 0.8, and 1.6. A temperature factor with values at 50, 100, and 120 degrees Celsius and an hours of operation factor at 5000, 10000 and 15000 hours is also defined. For example, the maximum peak AC current density for a 0.9 micron wide wire running for 5000 hours of operation at 120 degrees at 100 MHz is 10.125 mA/micron ($7.5 \times 1.8 \times 0.75$).

```
spacingTables (
  ( maxACCurrentDensity "Metal2"
    ( ( "frequency" nil nil "width" nil nil )
      'peak
      'tempFactor ( ( ( "temperature" ) )
        (
          50    2.2
          100   1.0
          120   0.75
        )
      )
      'hoursFactor ( ( ( "hours" ) )
        (
          5000  1.8
          10000 1.0
          15000 0.8
        )
      )
    )
  )
  4.0
)
(
  (100.0 0.3) 9.0
  (100.0 0.8) 7.5
  (100.0 1.6) 6.5
  (400.0 0.3) 7.5
  (400.0 0.8) 6.8
  (400.0 1.6) 6.0
)
) ;maxACCurrentDensity
) ;spacingTables
```

maxNumCorners

```
spacings (
    ( maxNumCorners tx_layer
      ['allCorner | 'concaveCorner | 'convexCorner]
      x_count
    ) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the maximum number of corners allowed in an island.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_count</i>	The maximum number of corners allowed.

Parameters

```
'allCorner | 'concaveCorner | 'convexCorner
```

The constraint applies to corners of this type.

Examples

The maximum number of concave corners allowed in a Metal1 island is 43.

```
spacings (
    ( maxNumCorners "Metal1"
      'concaveCorner
      43
    )
) ;spacings
```

minOneDArrayStructure

```
orderedSpacings (
    ( minOneDArrayStructure tx_cutLayer1 tx_metalLayer tx_cutLayer2
      'cutClass {f_width | (f_width f_length) | t_name}
      'numCuts x_numCuts
      'arraySpace f_arraySpacing
      'interViaSpace f_interViaSpacing
      'dualExtension (f_ext f_oppExt)
      f_cutSpacing
    )
) ;orderedSpacings
```

Specifies the minimum edge-to-edge spacing between cut shapes in a 1-D array on *cutLayer1*. The cut shapes in this 1-D array are aligned in the preferred routing direction specified for the metal layer below *cutLayer1*.

The minimum number of cut shapes in the 1-D array must be greater than or equal to *numCuts*. The edge-to-edge spacing of a cut shape in this 1-D array to any other cut shape on *cutLayer1* must be greater than or equal to *arraySpacing* and to a cut shape on *cutLayer2* must be greater than or equal to *interViaSpacing*, provided the parallel run length between the two cut shapes is greater than zero.

The cut shapes in the 1-D array must have *metalLayer* extensions greater than or equal to *ext* on one pair of opposite sides and *oppExt* on the other pair of opposite sides.

Values

<i>tx_cutLayer1</i>	The first cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_metalLayer</i>	The metal layer on which the constraint is applied (routing layer). Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer2</i>	The second cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_cutSpacing</i>	The edge-to-edge spacing between the cut shapes in the 1-D array must be greater than or equal to this value.

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'numCuts x_numCuts`

The number of cut shapes in a aligned 1-D array must be greater than or equal to this value.

`'arraySpace f_arraySpacing`

The spacing between a cut shape in the 1-D array and any other *cutLayer1* cut shape must be greater than or equal to this value if the parallel run length between the two shapes is greater than zero.

The 1-D array should be merged into a single rectangular box (with a grow or shrink operation) before applying the check, which should take precedence over other spacing constraints defined for the cut layer.

`'interViaSpace f_interViaSpacing`

The spacing between a cut shape in the 1-D array and a cut shape on *cutLayer2* must be greater than or equal to this value if the parallel run length between the two shapes is greater than zero.

`'dualExtension (f_ext f_oppExt)`

The cut shapes in the 1-D array must have *metalLayer* extensions equal to *ext* on one pair of opposite sides and *oppExt* on the other pair of opposite sides.

Example

The spacing between Via1 via cuts in a 1-D array with at least 6 via cuts, aligned in the horizontal direction, must be greater than or equal to 0.1. Each of these via cuts must have at least 0.03 and 0.05 Metal1 extensions on opposite edges.

Virtuoso Technology Data Constraints Reference

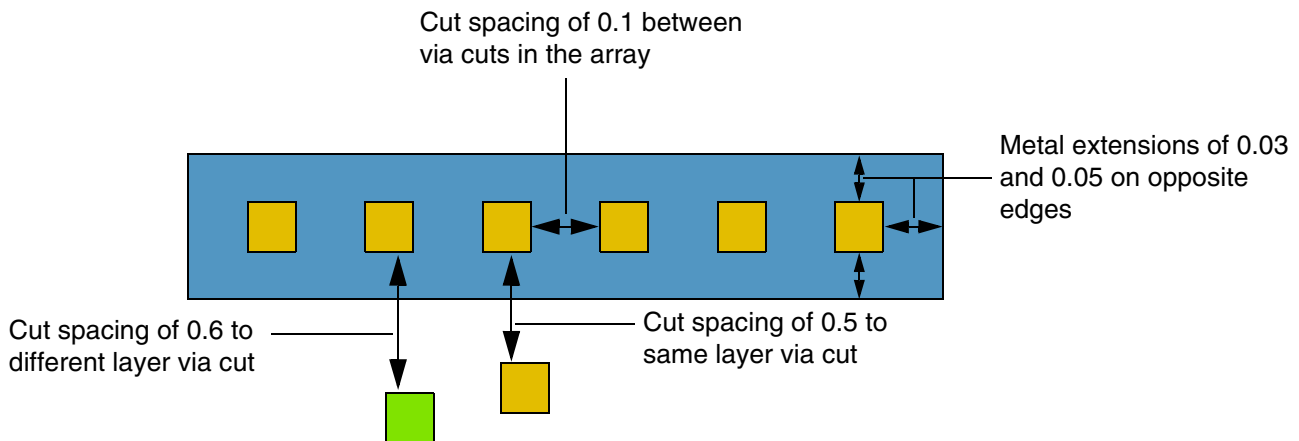
Miscellaneous Constraints

The spacing of a via cut in the 1-D array to any other Via1 via cut must be greater than or equal to 0.5 and to a Via2 via cut must be greater than or equal to 0.6, provided the two via cuts have parallel run length greater than zero.

```
orderedSpacings(  
  ( minOneDArrayStructure "Via1" "Metal1" "Via2"  
    'cutClass "VA"  
    'numCuts 6  
    'arraySpace 0.5  
    'interViaSpace 0.6  
    'dualExtension (0.03 0.05)  
    0.1  
  )  
);orderedSpacings
```

■ Via2
■ Via1
■ Metal1

Preferred routing direction on the metal layer below Via1: Horizontal



minStitchOverlap

```

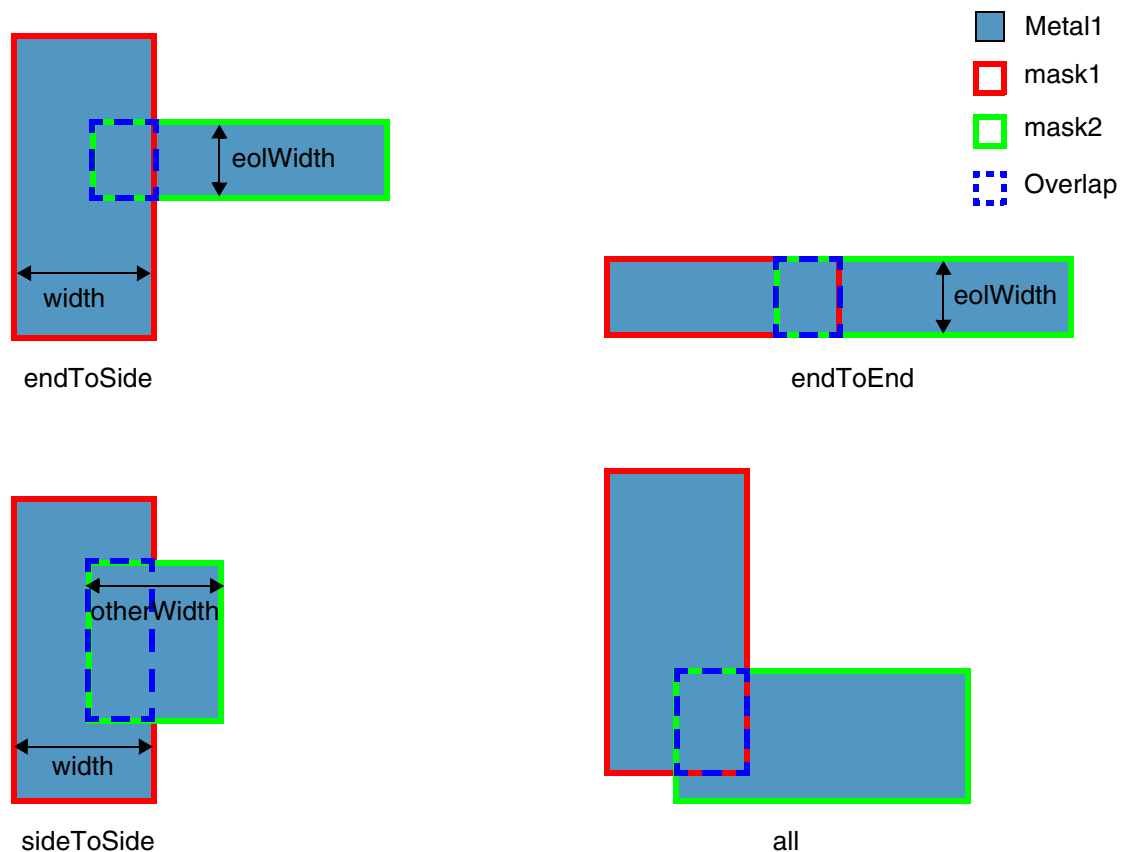
spacings (
    ( minStitchOverlap tx_layer
        ['all | 'endToEnd | 'endToSide | 'endToAny | 'sideToSide]
        ['eolWidth f_eolWidth]
        ['width f_width ['otherWidth f_otherWidth]]
        (f_overlapValue)
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum overlap between two shapes on the same layer, but on two different masks.

The figure below illustrates four different types of overlaps that can occur. Overlap type "all" includes all other overlap types; overlap type "endToAny" includes both "endToEnd" and "endToSide".

Note: Edges with width less than *eolWidth* are end-of-line edges. All other edges are considered as sides.



Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_overlapValue</i>	The overlap between the two shapes must be greater than or equal to this value. Type: Float

Parameters

<code>'all 'endToEnd 'endToSide 'endToAny 'sideToSide</code>	The overlap type. The default value is 'all. ■ 'all: The constraint applies to any two overlapping edges. ■ 'endToEnd: The constraint applies if the two overlapping edges are both end-of-line edges. ■ 'endToSide: The constraint applies if an end-of-line edge overlaps a side edge. ■ 'endToAny: The constraint applies if one of the overlapping edges is an end-of-line edge. ■ 'sideToSide: The constraint applies if the two overlapping edges are both side edges. Type: Boolean

<code>'eolWidth f_eolWidth</code>	The constraint applies only if the end-of-line width is less than this value. This parameter must be specified if overlap type is 'endToSide, 'endToEnd, or 'endToAny.

<code>'width f_width</code>	The constraint applies to shapes with width greater than or equal to this value. If 'eolWidth is also specified, the constraint applies only if the width of the shape without an end-of-line edge is greater than or equal to this value.

`'otherWidth f_otherWidth`

The constraint applies only if one shape has width greater than or equal to *width* and the other shape has width greater than or equal to this value.

This parameter is used with overlap types `'all` and `'sideToSide` when `'eolWidth` is not specified.

Examples

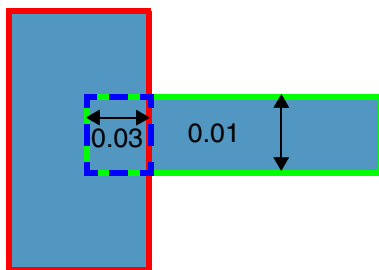
- Example 1: minStitchOverlap with endToSide and eolWidth
- Example 2: minStitchOverlap with endToEnd and eolWidth

Example 1: minStitchOverlap with endToSide and eolWidth

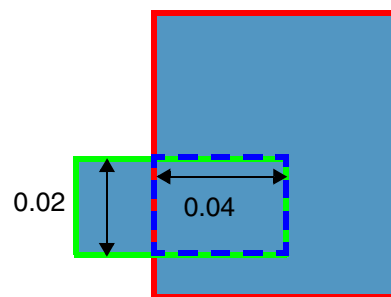
The end-to-side overlap between a Metal1 end-of-line edge less than 0.022 wide and a Metal1 side on different masks must be at least 0.04.

```
spacings (
  ( minStitchOverlap "Metal1"
    'endToSide
    'eolWidth 0.022
    (0.04)
  )
) ;spacings
```

■ Metal1
□ mask1
□ mask2
□ Overlap



a) FAIL. The end-of-line width is 0.01 (<0.022), but the overlap between the shapes is 0.03 (<0.04).



b) PASS. The end-of-line width is 0.02 (<0.022) and the overlap between the shapes is 0.04.

Virtuoso Technology Data Constraints Reference

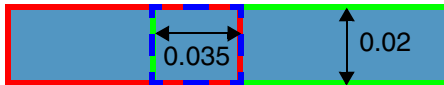
Miscellaneous Constraints

Example 2: *minStitchOverlap* with *endToEnd* and *eolWidth*

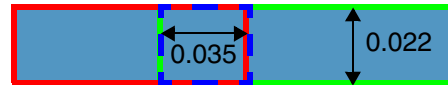
The end-to-end overlap between two Metal1 end-of-line edges less than 0.022 wide on different masks must be at least 0.04.

```
spacings(  
  ( minStitchOverlap "Metal1"  
    'endToEnd  
    'eolWidth 0.022  
    (0.04)  
  )  
); spacings
```

■ Metal1
□ mask1
□ mask2
□ Overlap



a) FAIL. The end-of-line width of the two shapes is 0.02 (<0.022), but the overlap is 0.035 (<0.04).



b) The constraint does not apply. The end-of-line width of the two shapes is equal to 0.022. For the constraint to apply, the end-of-line width must be less than 0.022.

msCrossTieLayers

```
orderedSpacings (  
    ( msCrossTieLayers tx_layer1 tx_layer2 )  
);orderedSpacings
```

Indicates that crosstie shields should be built for routes on the net on *layer1*, *layer2*, and the layers in between.

This can help minimize congestion.

Values

tx_layer1 tx_layer2

Crosstie shields should be built for routes on the net on *layer1*, *layer2*, and the layers in between.

Type: String (layer name) or Integer (layer number)

Parameters

None

Example

Crosstie shields should be built for routes on the net on Metal3, Metal4, Metal5 and Metal6.

```
orderedSpacings (  
    ( msCrossTieLayers "Metal3" "Metal6")  
);orderedSpacings
```

msLayerCrossTieInterval

```
spacings (
    ( msLayerCrossTieInterval tx_layer
      x_tracks
    )
) ;spacings
```

Defines how often crossties should be built on the specified layer. The constraint value specifies the interval as the number of tracks between crossties.

A value of 0 means that there are no tracks between crossties. A value of 1 results in a crosstie on every other track.

Values

<i>tx_layer</i>	The layer on which the crossties are built. Type: String (layer name) or Integer (layer number)
<i>x_tracks</i>	The number of tracks between crossties built on the specified layer.

Parameters

None

Example

The number of tracks between crossties built on Metal1 should be two.

```
spacings (
    ( msLayerCrossTieInterval "Metal1" 2 )
) ;spacings
```

msSameMask

```
spacings (
    ( msSameMask ['mask1 | 'mask2 | 'mask3] )
) ;spacings
```

Specifies that all shapes in a net group, to which the constraint is applied, must be on the same mask.

Optionally, you can specify the mask to be used.

Values

None

Parameters

'mask1 | 'mask2 | 'mask3

All shapes in the net group must be on the specified mask.

Type: Boolean

Example

All shapes in the net group must be on mask1.

```
spacings (
    ( msSameMask 'mask1 )
) ;spacings
```

msShieldingLimit

```
orderedSpacings (
    ( msShieldingLimit tx_layer1 tx_layer2 )
) ;orderedSpacings
```

Specifies a continuous set of layers to be used to build crossties.

Values

tx_layer1 tx_layer2

This layer pair and the layers in between define a continuous set of layers that can be used to build crossties.

Type: String (layer name) or Integer (layer number)

Parameters

None

Example

Crossties can be built only on layers Metal1, Metal2, and Metal3.

```
orderedSpacings (
    ( msShieldingLimit "Metal1" "Metal3")
) ;orderedSpacings
```

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

msShieldStyle

```
spacings (
    ( msShieldStyle x_style )
) ;spacings
```

Determines the type of shielding to route around a net.

Use a value listed in the table below to specify the shielding type.

Value	Name
0	none
1	parallel
2	below
3	belowParallel
4	above
5	aboveParallel
6	tandem
7	coaxial
8	split
9	splitParallel
10	splitBelow
11	splitBelowParallel
12	splitAbove
13	splitAboveParallel
14	splitAboveBelow
15	tandemSplit
16	via
17	parallelViaAlign
18	viaBelow
19	viaBelowParallel
20	viaAbove
21	viaAboveParallel

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Value	Name
22	viaAboveBelow
23	viaAboveBelowParallel
24	viaSplit
25	viaSplitParallel
26	viaSplitBelow
27	viaSplitBelowParallel
28	viaSplitAbove
29	viaSplitAboveParallel
30	viaSplitAboveBelow
31	tandemSplitViaAlign
128	crossTie

Values

x_style Encodes the shielding style for the net. Use a value listed in the table above to specify the shielding type. The values 32 through 127 are reserved.

Parameters

None

Example

The required shielding type is `crossTie` (value=128).

```
spacings(  
    ( msShieldStyle 128 )  
);spacings
```

multiMaskCheck

```
spacings(  
    ( multiMaskCheck tx_layer  
      x_value  
    )  
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the number of masks on a layer. The number of masks determines whether double or triple patterning checks are run on the layer.

Note: This constraint is expected to be specified at the design level.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_value</i>	The number of masks on the layer. If the value is 2, shapes on the layer are checked for odd cycle loops based on the spacing rules specified with the 'sameMask' parameter. If the value is 3, shapes on the layer are checked for wheel configuration violations based on the spacing rules specified with the 'sameMask' parameter. Note: Any integer value is allowed, but only 2 and 3 are supported.

Parameters

None

Example

The shapes on Metal3 must be checked for triple patterning errors and the shapes on Metal2 must be checked for double patterning errors.

```
spacings(  
    ( multiMaskCheck "Metal3" 3 )  
    ( multiMaskCheck "Metal2" 2 )  
) ;spacings
```

overlapViaGroup

```

spacings (
    ( overlapViaGroup tx_layer
        'groupName t_groupName
        'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
        'dualDistance (g_minDistance g_maxDistance)
        {'vertical | 'horizontal]
        ['sameMask]
        g_cuts
    )
) ;spacings

```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the maximum number of cuts in a via group, where the spacing between cuts is within a specified range.

Values

<i>tx_layer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_cuts</i>	The maximum number of cuts.

Parameters

'groupName *t_groupName*

Name of the via group.

'cutClass {*f_cutWidth* | (*f_cutWidth* *f_cutLength*) | *t_name*}

The constraint applies to the cut class specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_cutWidth*: Width
- *f_cutLength*: Length
- *t_name*: Name of the cut class

'dualDistance (*g_minDistance* *g_maxDistance*)

Any two cuts in the via group must have a spacing of more than *g_minDistance* and less than *g_maxDistance*.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

'vertical | 'horizontal

The constraint applies only in the specified direction on spacing between two cuts whose parallel run length is more than 0 in the orthogonal direction.

'sameMask

The constraint applies only to via cuts on the same mask.

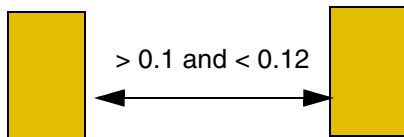
Type: Boolean

Example

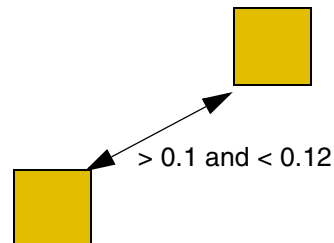
The maximum number of cuts for cutClass VA in viaGroup VGA is 3. The cuts in the group need to meet the dualDistance range.

```
spacings (
  (overlapViaGroup "Via1"
    'groupName "VGA"
    'cutClass "VA"
    'dualDistance (0.1 0.12)
    'vertical
    3)
  )
) ;spacings
```

■ Via1



Can form a group because the vias have parallel overlap.



Cannot form a group because the vias do not have parallel overlap.

pinConnectBlockage

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_size</i>	The area of the blockage. Type: Float, specified in microns.

Parameters

<code>'enclosedLength</code> <i>f_enclosedLength</i>	Specifies that any enclosed empty area (that is, a donut hole formed by the metal) would have a length along the pin edge greater than or equal to the given value if it overlaps with blockage area.

Example

Specifies a blockage area of 0.06 for the M8 layer within an enclosed length of 0.09.

```
spacings(  
  ( pinConnectBlockage "M8"  
    'enclosedLength 0.09  
    0.06  
  )  
) ;spacings
```

preColoredLayers

```

spacings (
    ( preColoredLayers ( tx_layer1 | (tx_layer1 tx_purpose1) )
      [ tx_layer2 | (tx_layer1 tx_purpose2) ] ...
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Lists precolored layers or layer-purpose pairs in the `VirtuosoMPTSetup` constraint group. The shapes on such layers and purposes are colorable but are neither colored by the coloring engine nor flagged as errors by the checker. If required, you can specify a color for these layers in the palette.

Values

<code>tx_layer1</code>	The layer on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<code>tx_layer1 tx_purpose1</code>	The layer-purpose pair on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer and purpose numbers)

Parameters

None

Example

The layer `Via1` with purpose `fill` is indicated as precolored.

```

constraintGroups (
    ( "virtuosoDefaultMPTSetup" nil "virtuosoMPTSetup"
      spacings (
        ( layerDefaultColor "Metal1" "mask2")
        ( layerDefaultColor "Metal2" "mask2")
        ( preColoredLayers (Via1 "fill") )
      ) ;spacings
      interconnect (
        ( validLayers (Metal1 Via1 Metal2 Via2 Metal3) )
        ( validPurposes 'include ("fill" "drawing") )
        ( excludeLPPs (Metal1 "fill") (Metal3) )
      )
    )
)

```

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

rectShapeDir

```
spacings (
  ( rectShapeDir tx_layer
    ['extendBy f_extendBy ['widthRanges (g_ranges)]]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ]
  [['exceptViaLayer tx_viaLayer
    'exceptViaSize (f_viaSize1 f_viaSize2)
    'exceptExactSize (f_exactMetalSize1 f_exactMetalSize2)]
    | 'exceptViaEdgeAligned | 'exceptVia
  ]
  ['exceptEdgeLength f_maxLength 'width f_maxWidth ['twoSides]]
  ['exceptRanges (g_exceptRanges)]
  ['exceptNonCorePins]
  ['exceptWidth f_width {'widthHorizontal | 'widthVertical}]
  ['partialTracks]
  ['onlyOne]
  {"any" | "horizontal" | "vertical"}
)

) ;spacings

spacingTables (
  ( rectShapeDir tx_layer
    (( "width" nil nil )
    ['extendBy f_extendBy ['widthRanges (g_ranges)]]
    [['exceptViaLayer tx_viaLayer
      'exceptViaSize (f_viaSize1 f_viaSize2)
      'exceptExactSize (f_exactMetalSize1 f_exactMetalSize2)
      | 'exceptViaEdgeAligned | 'exceptVia
    ]
    ]
    ['exceptEdgeLength f_maxLength 'width f_maxWidth ['twoSides]]
    ['exceptRanges (g_exceptRanges)]
    ['exceptNonCorePins]
    ['exceptWidth f_width {'widthHorizontal | 'widthVertical}]
    ['partialTracks]
    ['onlyOne]
    [ "any" | "horizontal" | "vertical" ]
  )
  (g_table)
)

) ;spacingTables
```


Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Specifies that the shapes on a layer must be rectangular. Depending on their width, these shapes can be restricted to a certain direction. A square shape is both horizontal and vertical. Optionally, narrow rectangular shapes can have small jogs.

Additionally, this constraint is used with the `allowedWidthRanges` constraint to specify a set of discrete widths for a layer. The `allowedLengthRanges` constraint must also specify a minimum length greater than or equal to the minimum width to avoid ambiguity while determining the direction.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>"width" nil nil</code>	This identifies the index for <code>table</code> .
<code>g_table</code>	The <code>g_table</code> row is defined as: <code>f_width {"any" "horizontal" "vertical"}</code> where, <code>f_width</code> is the width of the rectangle. Type: A 1-D table specifying floating-point width values and the direction of the rectangle

Parameters

<code>'extendBy f_extendBy</code>	(Virtuoso Advanced Node for Layout Standard) All end-of-line edges must be extended by this value before the constraint is applied.
<code>'widthRanges g_ranges</code>	(Virtuoso Advanced Node for Layout Standard) The <code>'extendBy</code> parameter applies only to the end-of-line edges that have width in this range. Type: Floating-point values specifying a <u>range</u> of widths

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

(Virtuoso Advanced Node for Layout Standard) Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Standard) The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Advanced Node for Layout Standard) The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

```
'exceptViaLayer tx_viaLayer  
'exceptViaSize f_viaSize1 f_viaSize2  
'exceptExactSize f_exactMetalSize1 f_exactMetalSize2  
'exceptViaEdgeAligned  
'exceptVia
```

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply to a metal shape with dimensions *exactMetalSize1* and *exactMetalSize2* if a via with dimensions *viaSize1* and *viaSize2* placed on *viaLayer* overlaps it. The metal shape left after subtracting such a shape must either be a rectangle or multiple exactly aligned rectangles.

Moreover, this exemption applies only if an edge of this metal shape in the direction of the constraint is exactly aligned with an edge of such rectangles.

(Virtuoso Layout Suite EXL and higher tiers) The *exceptViaEdgeAligned* parameter specifies that the rectangular shape checking is exempted by a via, which has an edge aligned to the metal edge of a connected wire.

Type: String (layer name) or Integer (layer number); Float

(Virtuoso Layout Suite EXL and higher tiers) The *exceptVia* parameter specifies that the rectangular shape checking is exempted by a via.

```
'exceptEdgeLength f_maxLength
```

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if the rectangular shape has a jog with width less than or equal to this value.

```
'width f_maxWidth
```

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if the width of the shape with the *maxLength* wide jog is less than or equal to this value.

```
'twoSides
```

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if the rectangular shape has *maxLength* wide jogs on both sides. Additionally, the two jogs must not have a vertical offset.

```
'exceptRanges (g_exceptRanges)
```

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

The constraint does not apply if a rectangle has width in this range.

Type: Floating-point values specifying a range of widths that are exempted

`'exceptNonCorePins` (Virtuoso Advanced Node for Layout Standard) The constraint does not apply to the connections to non-standard cell pins. In other words, the connection of a default-width wire to a wide I/O pad or a block pin is allowed.

Type: Boolean

`'exceptWidth f_width {'widthHorizontal | 'widthVertical}`

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply if the width in the direction specified by `widthHorizontal` or `widthVertical` is less than or equal to `f_width`.

`'partialTracks` (Virtuoso Layout Suite EXL and higher tiers) The rectangular shape checking applies only to specific tracks.

Type: Boolean

`'onlyOne` The constraint allows only one shape on the specified layer.

`"any" | "horizontal" | "vertical"`

The direction of the rectangle.

Type: Boolean

Examples

- Example 1: rectShapeDir with vertical
- Example 2: rectShapeDir with exceptViaLayer, exceptViaSize, and exceptExactSize
- Example 3: rectShapeDir with extendBy and widthRanges
- Example 4: rectShapeDir with exceptEdgeLength, width, and twoSides
- Example 5: rectShapeDir with exceptRanges

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Example 1: rectShapeDir with vertical

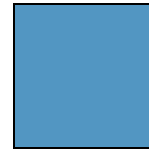
A shape on layer Metal1 must be a rectangle in the vertical direction.

```
spacings(  
  ( rectShapeDir "Metal1"  
    "vertical"  
  )  
) ;spacings
```

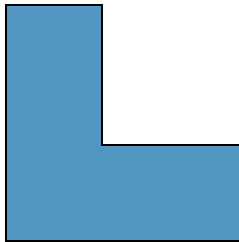
 Metal1



a) PASS. The shape is a vertical rectangle.



b) PASS. The shape is a square and a square is both horizontal and vertical.



c) FAIL. The shape is not a rectangle.



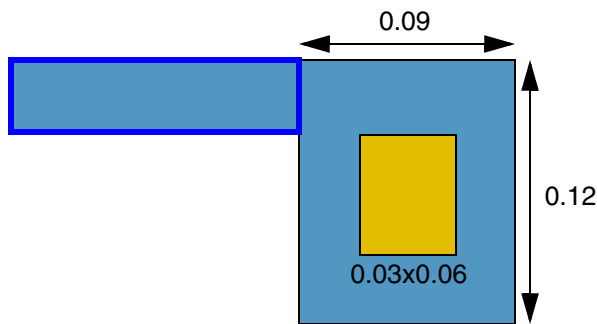
d) FAIL. The shape is a horizontal rectangle.

Example 2: rectShapeDir with exceptViaLayer, exceptViaSize, and exceptExactSize

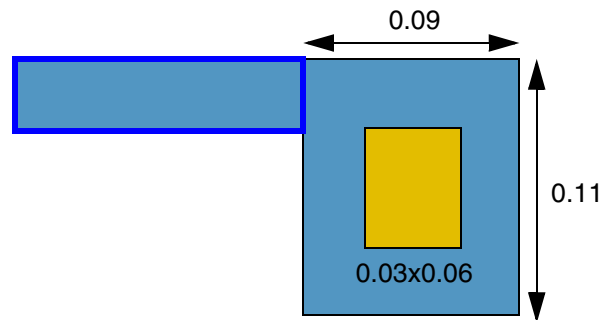
A Metal1 shape with dimensions 0.09×0.12 is overlapped by a 0.03×0.06 via. The constraint is met if the remaining portion of the metal shape is a horizontal rectangle, exactly aligned with a horizontal edge of the exempted portion of the metal shape.

```
spacings (
  ( rectShapeDir "Metal1"
    'exceptViaLayer "Via1"
    'exceptViaSize (0.03 0.06)
    'exceptExactSize (0.09 0.12)
    "horizontal"
  )
) ; spacings
```

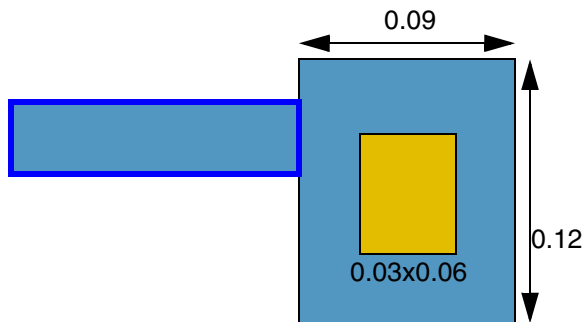
■ Via1
■ Metal1



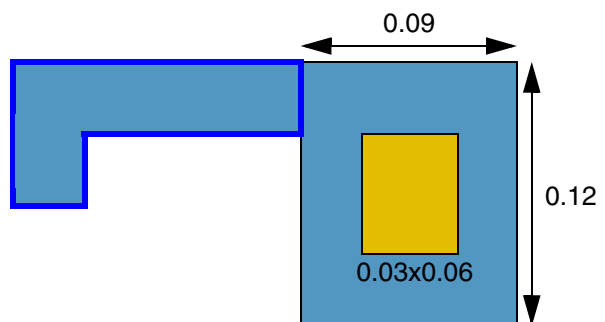
a) PASS. The 0.09×0.12 metal shape is overlapped by a 0.03×0.06 via on layer Via1, and is, therefore, exempt. The remaining metal shape (with a thick blue border) is a horizontal rectangle.



b) FAIL. The 0.09×0.11 metal shape does not meet the 'exceptExactSize requirement ($0.11 < 0.12$).



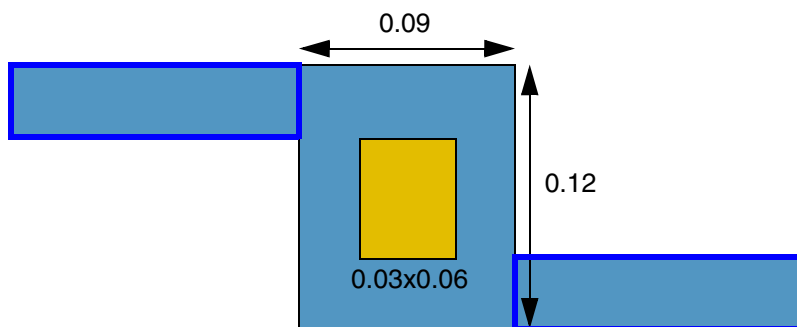
c) FAIL. The 0.09×0.12 metal shape is overlapped by a 0.03×0.06 via on layer Via1, and is, therefore, exempt. The remaining metal shape (with a thick blue border) is a horizontal rectangle, but is not exactly aligned with a horizontal edge of the exempted portion of the metal shape.



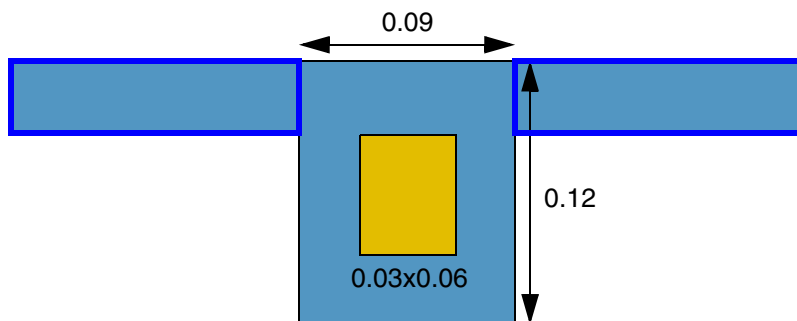
d) FAIL. The 0.09×0.12 metal shape is overlapped by a 0.03×0.06 via on layer Via1, and is, therefore, exempt. However, the remaining metal shape (with a thick blue border) is not a rectangle.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints



e) FAIL. The 0.09x0.12 metal shape is overlapped by a 0.03x0.06 via on layer Via1, and is, therefore, exempt, and the remaining metal shapes (with a thick blue border) are both rectangles. However, the two rectangles are not exactly aligned with the same horizontal edge of exempted portion of the metal shape.



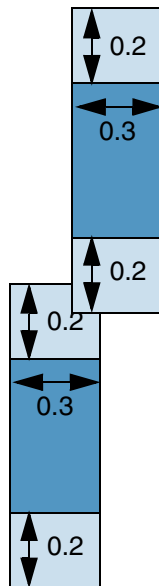
f) PASS: The 0.09x0.12 metal shape is overlapped by a 0.03x0.06 via on layer Via1, and is, therefore, exempt. The remaining metal shapes (with a thick blue border) are both rectangles and are exactly aligned with the same horizontal edge of exempted portion of the metal shape.

Example 3: rectShapeDir with extendBy and widthRanges

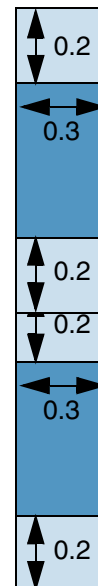
When the edges of a rectangular `Metal1` shape with width less than or equal to `0.3` or equal to `0.6` or `0.8` wide are extended by `0.2`, the resulting shape must also be rectangular.

```
spacings (
  ( rectShapeDir "Metal1"
    'extendBy 0.2
    'widthRanges ("<=0.3" "0.6" "0.8")
    "any"
  )
) ; spacings
```

Metal1 extension
 Metal1



a) FAIL. Both shapes have end-of-line edges in one of the width ranges (≤ 0.3), so each of these edges is extended by 0.2. However, the resulting shape is not a rectangle.



b) PASS. Both shapes have end-of-line edges in one of the width ranges (≤ 0.3), so each of these edges is extended by 0.2. The resulting shape is a rectangle.

Virtuoso Technology Data Constraints Reference

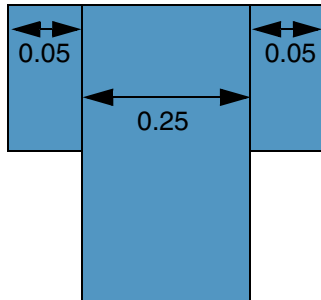
Miscellaneous Constraints

Example 4: *rectShapeDir* with *exceptEdgeLength*, *width*, and *twoSides*

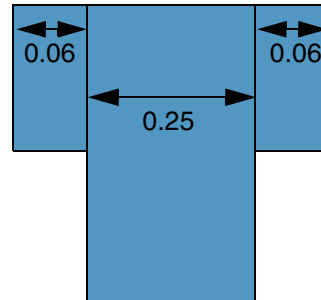
A vertical rectangle on layer `Metal1` with width `0.25` must have `0.05` wide jogs on both sides.

```
spacings(  
  ( rectShapeDir "Metal1"  
    'exceptEdgeLength 0.05 'width 0.25 'twoSides  
    "vertical"  
  )  
) ;spacings
```

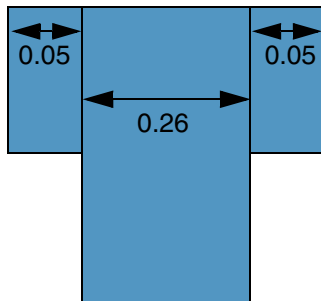
■ Metal1



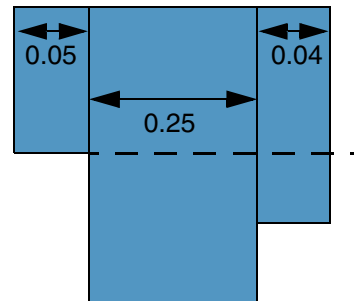
a) PASS. The two jogs are each 0.05 wide and the width of the vertical rectangle is 0.25.



b) FAIL. The width of the vertical rectangle is 0.25, but the two jogs are each 0.06 wide (>0.05).



c) FAIL. The two jogs are each 0.05 wide, but the width of the vertical rectangle is 0.26 (>0.25).



d) FAIL. The width of the vertical rectangle is 0.25, but one of the jogs is 0.04 wide (<0.05). Additionally, the two jogs are vertically offset (one jog is longer than the other).

Example 5: rectShapeDir with exceptRanges

All rectangles on layer `Metal1` must be vertical. However, the constraint does not apply if a rectangle has width in one of the following ranges:

- Greater than or equal to `0.5` and less than or equal to `0.08`
- Greater than `0.1` and less than or equal to `0.12`

```
spacings (
  ( rectShapeDir "Metal1"
    'exceptRanges ("[0.5 0.8]" "(0.1 0.12]")
    "vertical"
  )
) ;spacings
```

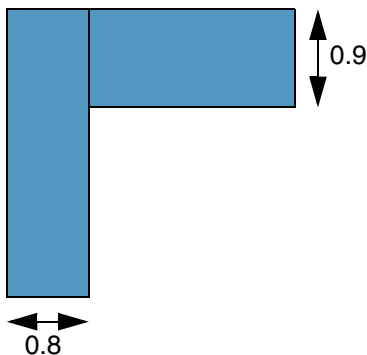
■ Metal1



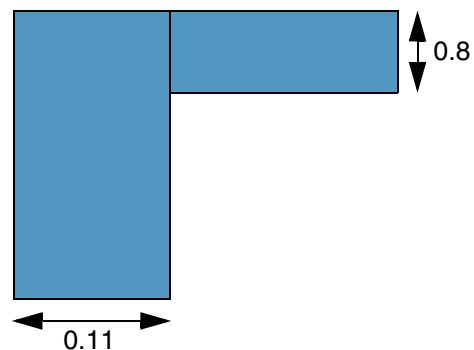
a) FAIL. The constraint applies because the width of the rectangle is not in the exempted width range. However, the rectangle is horizontal.



b) The constraint does not apply because the width of the rectangle is in the exempted width range.



c) FAIL. The constraint applies to the rectangle that is 0.9 wide because width 0.9 is not in the exempted width range. However, this rectangle is horizontal.



d) The constraint does not apply because the widths 0.8 and 0.11 are both in the exempted width ranges.

sameMaskOnLayer

```
spacings (
    ( sameMaskOnLayer tx_layer
      ['mask1 | 'mask2 | 'mask3 | 'mask4]
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the mask type for all shapes owned by the net associated with this constraint on a layer.

Note: This constraint is expected to be specified at the design level.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

<code>'mask1 'mask2 'mask3 'mask4</code>	All shapes on the specified layer must be of this mask type. Type: Boolean

Example

All shapes on layer Metal1 owned by the net for which the `sameMaskOnLayer` constraint is defined must be on mask1.

```
spacings (
    ( sameMaskOnLayer "Metal1"
      'mask1
    )
) ;spacings
```

trimMetalTrack

```

spacings (
    ( trimMetalTrack tx_trimLayer
      'coreOffset f_offset
      ['mask1 | 'mask2 | 'mask3]
      ['onTrack ['halfInFiller | 'halfNonMerged]
        ['includeLineEnd ['trimCenter]]]
      ['metalTrackOffset f_metalTrackNumber
        'metalTrackPitch f_metalTrackPitch]
      ['groupName f_groupName]
      f_pitch
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Specifies the pitch of the tracks on *trimLayer*. The direction of the tracks is orthogonal to the direction of the routing layer that *trimLayer* trims.

Values

<i>tx_trimLayer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_pitch</i>	The pitch value.

Parameters

'coreOffset <i>f_offset</i>	The starting coordinate of the track with respect to the origin of the core.
'mask1 'mask2 'mask3	The mask the trim metal grid is applied on. Type: Boolean
'onTrack	All inserted trims must be on the specified track.
'halfInFiller	(Virtuoso Layout Suite EXL and higher tiers) Trim shapes on certain application-defined filler cells that are on a half track are exempted.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

<code>'halfNonMerged</code>	(Virtuoso Layout Suite EXL and higher tiers) Non-merged trim shapes that are on half track are exempted.
<code>'includeLineEnd</code>	(Virtuoso Layout Suite EXL and higher tiers) The line-ends which do not have a trim, snap to the grid.
<code>'trimCenter</code>	(Virtuoso Layout Suite EXL and higher tiers) The line-end edges align to the center of the trim grid.
<code>'metalTrackOffset <i>f_metalTrackNumber</i> 'metalTrackPitch <i>f_metalTrackPitch</i></code>	The metal track number that the trim metal track is applied on (<i>f_metalTrackNumber</i>) and the metal track pitch (<i>f_metalTrackPitch</i>) at which the trim metal track should be repeated. Type: Integer
<code>'groupName <i>f_groupName</i></code>	(Virtuoso Layout Suite EXL and higher tiers) The group name of the track definition.

Examples

- Example 1: trimMetalTrack with mask
- Example 2: trimMetalTrack with metalTrackOffset and metalTrackPitch

Example 1: trimMetalTrack with mask

The tracks on TM1 are defined on mask1, with the starting coordinate of 0.01 with respect to the origin of the core and a repeatable pitch of 0.2. If TM1 trims Metal1 and Metal1 is routed in the vertical direction, the direction of the tracks on TM1 would be horizontal.

```
spacings(  
  ( trimMetalTrack "TM1"  
    'coreOffset 0.01  
    'mask1  
    0.2  
  )  
) ;spacings
```

Example 2: trimMetalTrack with metalTrackOffset and metalTrackPitch

Three vertical trim metal grids are defined: TM1, TM2, TM3. The TM1 trim grid applies to horizontal metal tracks 1, 4, 7, and so on. The TM2 trim grid applies to horizontal metal tracks 2, 5, 8, and so on. The TM3 trim grid applies to horizontal metal tracks 3, 6, 9, and so on.

```

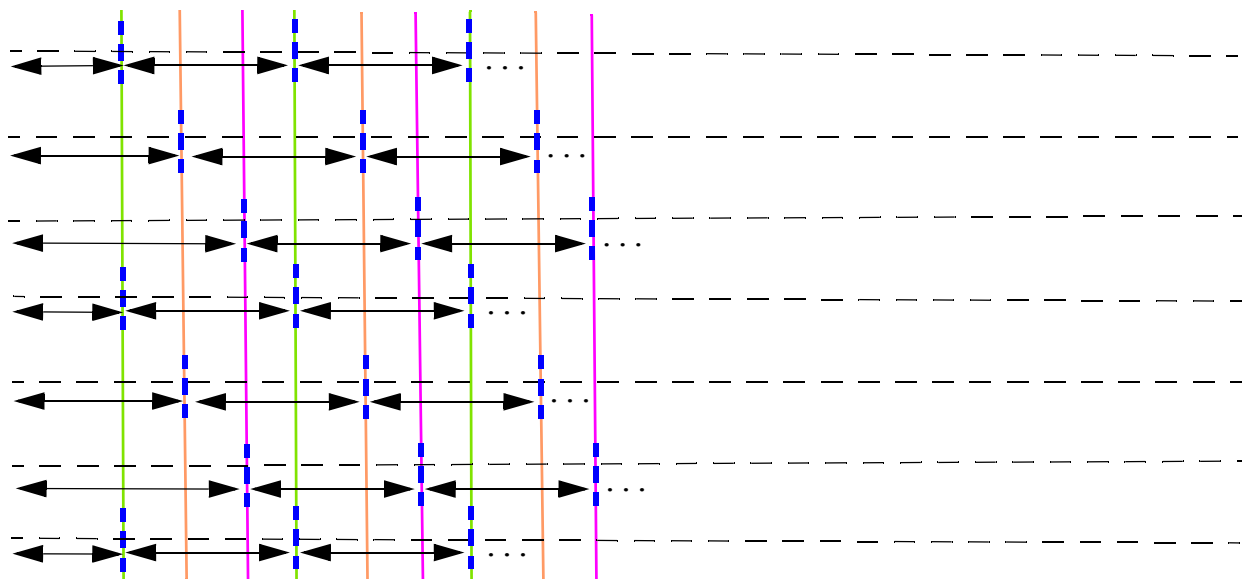
spacings (
  (trimMetalTrack "TM1"
    'coreOffset 0.01
    'metalTrackOffset 1
    'metalTrackPitch 3
    0.2)
  (trimMetalTrack "TM2"
    'coreOffset 0.03
    'metalTrackOffset 2
    'metalTrackPitch 3
    0.2)
  (trimMetalTrack "TM3"
    'coreOffset 0.06
    'metalTrackOffset 3
    'metalTrackPitch 3
    0.2)
) ;spacings

```

TM1

TM2

TM3



trimShape

```

spacings (
    ( trimShape tx_layer
        'exactWidth f_exactWidth
        ['maxLength f_maxLength]
        ['exceptSpacing f_spacing]
        ['exceptWidth f_width]
        ['gapMerged g_gap]
        ['rowBoundaryMaxLength f_rowBoundaryMaxLength]
        ['extension f_extension ['metalEdge]]
        ['mask1 | 'mask2 | 'mask3 | 'mask4]
        ['metalMask1 | 'metalMask2 | 'metalMask3 | 'metalMask4]
        ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['adjacentWidthExtension (f_adjWidth f_adjExtension)]
    ['keepoutRegion (f_offset1 f_offset2)]
    {"adjacentTrack" | "midTrack" | "extension"}
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) 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.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"adjacentTrack" "midTrack" "extension"	

The extension model.

- `adjacentTrack`: Shapes extend to the centerline of adjacent tracks.
- `midTrack`: Shapes extend by half of the required spacing of the wires.
- `extension`: Shapes extend by the specified value (*f_{extension}*) on either side from the center.

Shapes can be merged if they are aligned.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Parameters

`'exactWidth f_exactWidth`

The width of the shape formed on the trim metal layer must be equal to this value.

`'maxLength f_maxLength`

The length of the shape formed on the trim metal layer must be less than or equal to this value.

`'exceptSpacing f_spacing`

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.

`'exceptWidth f_width`

No shape is formed on the trim metal layer if the width of the line-end of a wire is greater than this value.

`'gapMerged g_gap`

(Virtuoso Layout Suite EXL and higher tiers) Perfectly aligned trim shapes with gap length less than or equal to the given value are merged.

`'rowBoundaryMaxLength f_rowBoundaryMaxLength`

(Virtuoso Layout Suite EXL and higher tiers) Specifies a different trim max length value when trims overlap with any cell row boundaries.

`'extension f_extension`

The trim metal shape must extend by this value on either side from the center of the metal that it trims. This parameter is used only when the constraint value is "extension".

`'metalEdge`

The trim metal shape must extend from the edges of the metal that it trims.

Type: Boolean

`'mask1 | 'mask2 | 'mask3 | 'mask4`

The mask of the trim layer to which the constraint is applied. This is used only for multi-patterned trim layers.

Type: Boolean

`'metalMask1 | 'metalMask2 | 'metalMask3 | 'metalMask4`

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

(Virtuoso Layout Suite EXL and higher tiers) The mask of the trimmed metal layer to which the constraint is applied.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'keepoutRegion (f_offset1 f_offset2)
```

Applicable only when *extension* is used.

The *f_offset1* value specifies the keepout offset in the metal direction. The *f_offset2* value specifies the keepout offset in the direction perpendicular to the metal layer.

If no metal shape is present in the keepout region, then the extension of the trim shape over the metal should be greater than or equal to the defined *f_extension* value. The metal is considered to be outside the keepout region if it only touches the keepout region and is not partly within it.

```
'adjacentWidthExtension (f_adjWidth f_adjExtension)
```

Applicable only when `adjacentTrack` is used. It specifies the width of the adjacent track and the extension of the trim shape from the metal edges. When the width of the adjacent track is greater than or equal to $f\_adjWidth$, the required trim metal extension is $f\_adjExtension$.

Examples

- [Example 1: trimShape with midTrack](#)
- [Example 2: trimShape with adjacentTrack](#)
- [Example 3: trimShape with extension](#)
- [Example 4: trimShape with extension and keepoutRegion](#)
- [Example 5: trimShape with extension and metalEdge](#)
- [Example 6: trimShape with adjacentWidthExtension](#)

Virtuoso Technology Data Constraints Reference

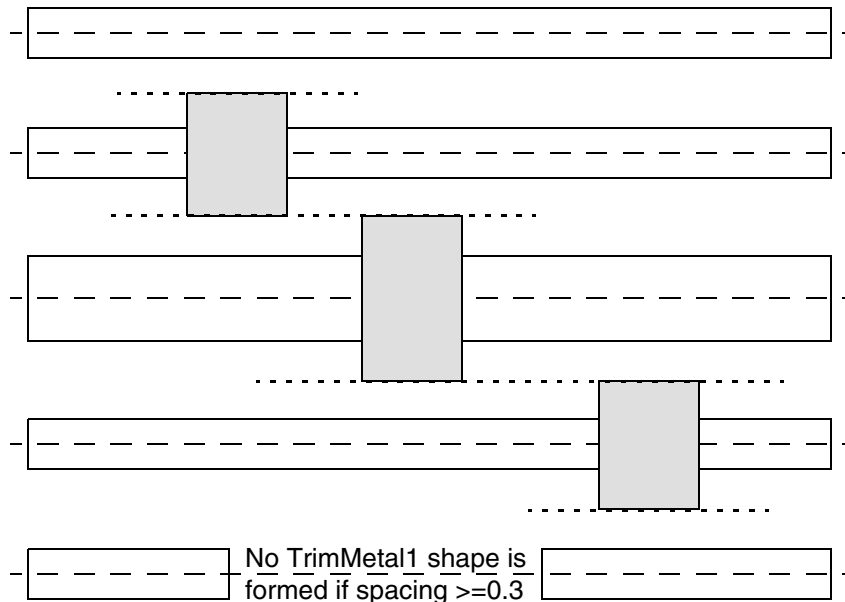
Miscellaneous Constraints

Example 1: trimShape with midTrack

The width of the TrimMetal1 shapes must be equal to 0.2 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 0.3.

```
spacings (
  ( trimShape "TrimMetal1"
    'exactWidth 0.2
    'exceptSpacing 0.3
    "midTrack"
  )
) ;spacings
```

□ TrimMetal1

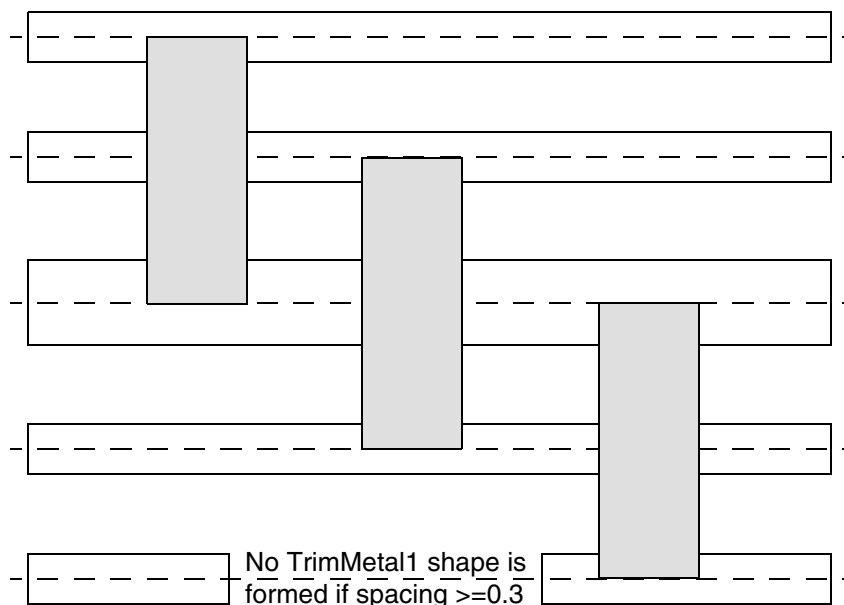


Example 2: trimShape with adjacentTrack

The width of each TrimMetal1 shape must be equal to 0.2 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 0.3.

```
spacings (
  ( trimShape "TrimMetal1"
    'exactWidth 0.2
    'exceptSpacing 0.3
    "adjacentTrack"
  )
) ;spacings
```

□ TrimMetal1

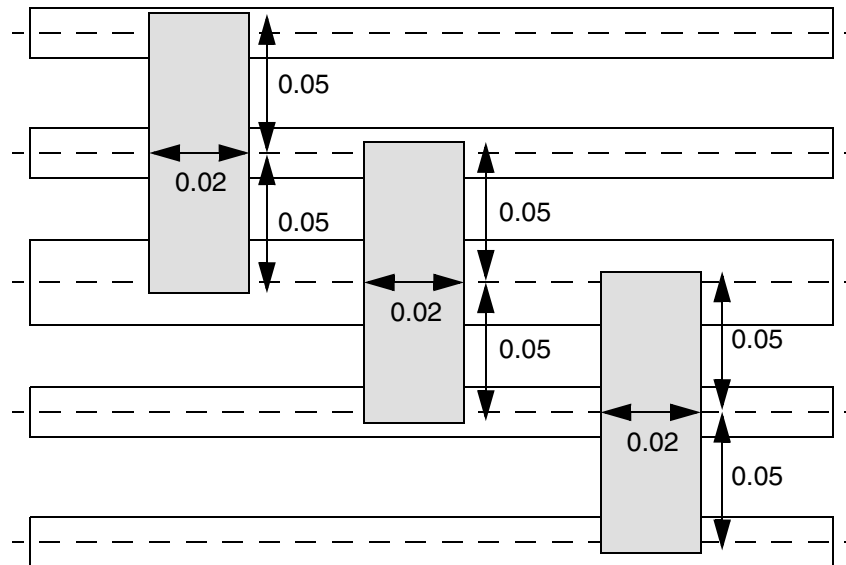


Example 3: trimShape with extension

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.

```
spacings (
  ( trimShape "TrimMetal1"
    'exactWidth 0.02
    'extension 0.05
    "extension"
  )
) ; spacings
```

□ TrimMetal1



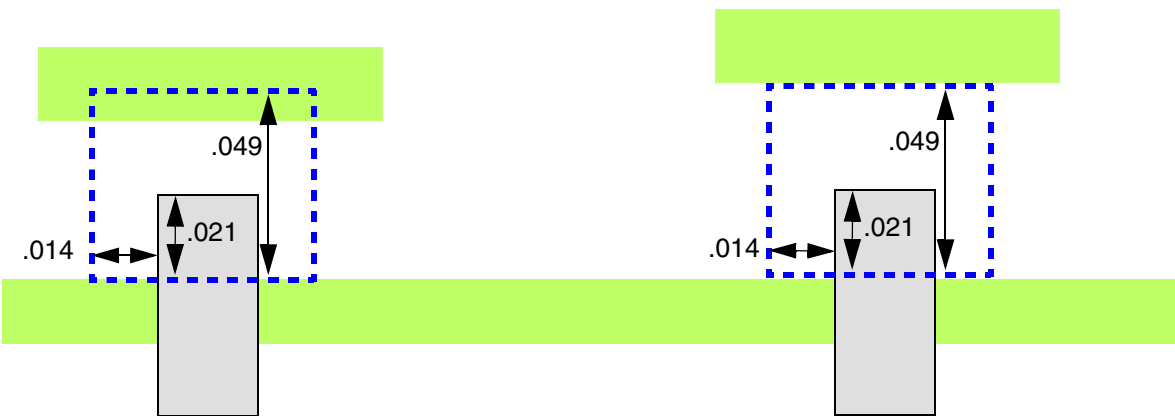
Example 4: trimShape with extension and keepoutRegion

The following trimShape constraints are defined in AND groups for TrimMetal1 shapes:

```
1. spacings(
      ( trimShape "TrimMetal1"
        'edgeExtension 0.021 "extension"
      )
) ;spacings
```

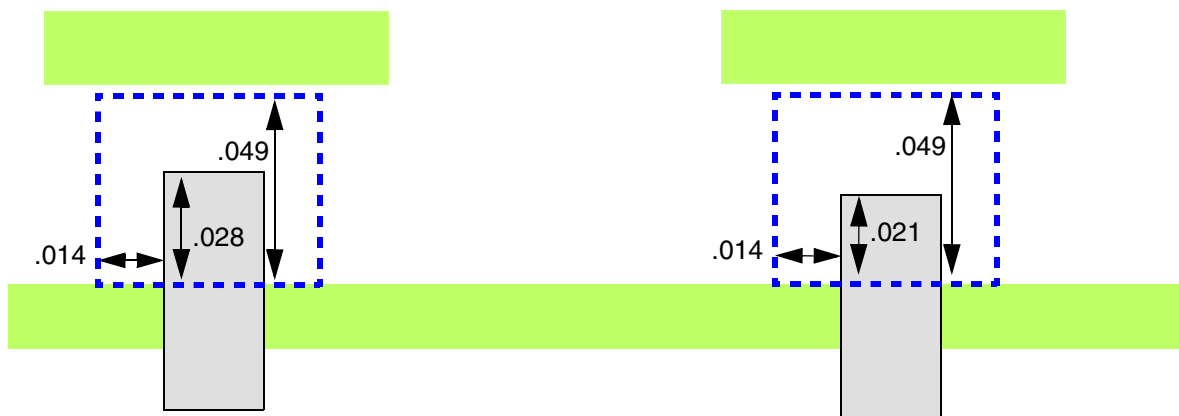
- TrimMetal1
- Trim
- extension

```
2. spacings(
    ( trimShape "TrimMetal1" 'keepoutRegion (0.014 0.049)
      'edgeExtension 0.028 "extension"
    )
) ;spacings
```



a. Pass. Rule 1 is met. Rule 2 is not applicable because the metal intersects the keepout region.

b. Fail. Rule 1 is met. Rule 2 is applicable but not met because the metal touches the keepout region and extension must be more than or equal to .028.



c. Pass. Rule 1 is met. Rule 2 is also met because no metal intersects the keepout region and the extension is more than or equal to .028.

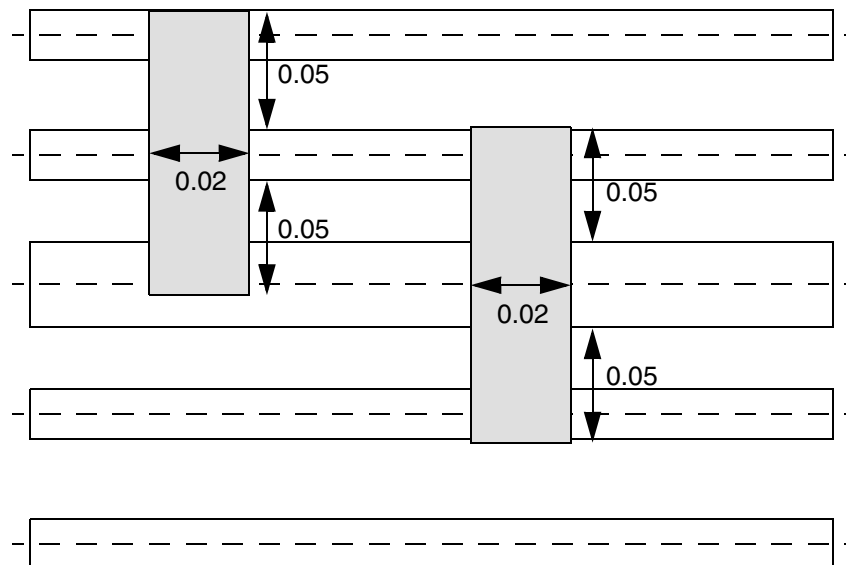
d. Fail. Rule 1 is met. Rule 2 is violated because no metal intersects the keepout region and the extension must be more than or equal to .028.

Example 5: trimShape with extension and metalEdge

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.

```
spacings (
  ( trimShape "TrimMetal1"
    'exactWidth 0.02
    'extension 0.05 'metalEdge
    "extension"
  )
) ;spacings
```

□ TrimMetal1



Example 6: trimShape with adjacentWidthExtension

Trim T1 and T3 snap to the center of the adjacent track because the width of the adjacent track is less than 0.09. The lower edge of trim T2 snaps to the center of the adjacent track. However, the upper edge of Trim T2 extends by 0.02 because the width of the adjacent top

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

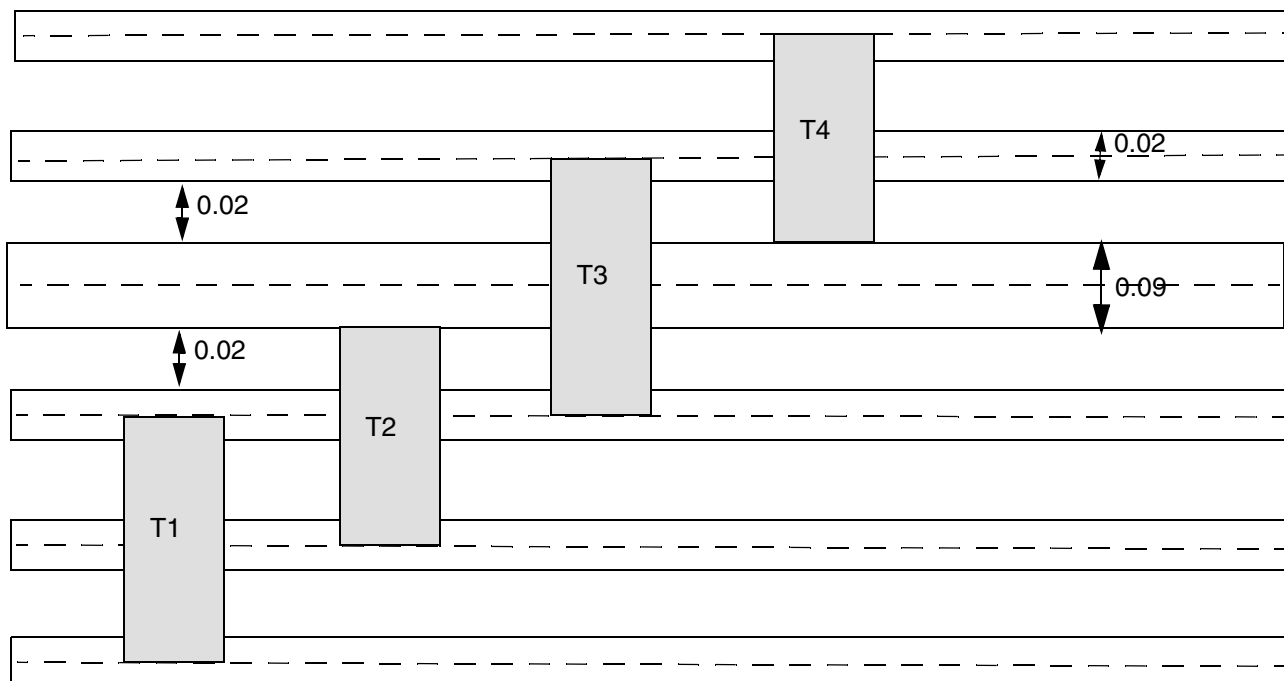
track is ≥ 0.09 . Similarly, the lower edge of trim T4 extends by 0.02 and the upper edge snaps to the center of the adjacent track.

```

spacings(
  ( trimShape
    'exactWidth      0.1
    'adjacentWidthExtension (0.09 0.02)
    "adjacentTrack"
  )
) ;spacings

```

TrimMetal1



useExistingShapesAsShield

```
spacings (
    ( useExistingShapesAsShield )
) ; spacings
```

Controls whether existing power or ground shapes can be used as shields.

Values

None

Parameters

None

validCutClass

```
spacings(  
    ( validCutClass tx_cutLayer  
      ['rowCol (x_row x_col)']  
      {(f_width f_length) | t_name}  
    )  
); spacings
```

Specifies the cut class that is allowed on the given layer.

This constraint should be used in a net's default constraint group, and not in the `foundry` constraint group.

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>(f_width f_length) t_name</i>	The cut class allowed for the specified layer, specified by width and length or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class

Parameters

```
'rowCol (x_row x_col)
```

The number of rows and columns in a multiple-cut via of the specified cut class must be equal to *row* and *col*, respectively.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Example

The cuts in a 0.3x0.3 via on layer Via1 must be organized into 2x2 array.

```
spacings(  
  ( validCutClass "Via1"  
    'rowCol (2 2)  
    (0.3 0.3)  
  )  
) ;spacings
```

vertexInsideForbidden

```
orderedSpacings(  
    ( vertexInsideForbidden tx_layer1 tx_layer2 )  
);orderedSpacings
```

(Virtuoso Advanced Node for Layout Only) Specifies that the vertices of a shape on *layer1* are not allowed inside a shape on *layer2*.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

None

Virtuoso Technology Data Constraints Reference

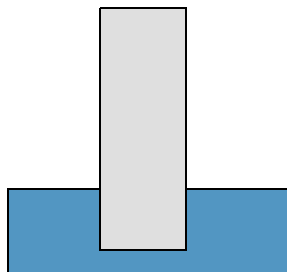
Miscellaneous Constraints

Example

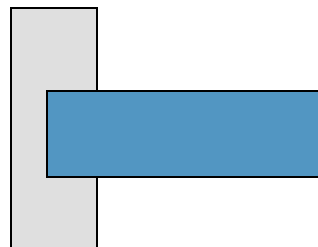
The constraint allows Metal2 vertices inside a Metal1 shape. However, Metal1 vertices are not allowed inside a Metal2 shape.

```
orderedSpacings(  
  ( vertexInsideForbidden "Metal1" "Metal2" )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



PASS. Metal2 vertices are allowed inside a Metal1 shape.



FAIL. Metal1 vertices are not allowed inside a Metal2 shape.

voltageMode

```
spacings (
    ( voltageMode
        ["maxDelta" | "absolute"]
    )
) ;spacings
```

Specifies the method of calculating the voltage of a net.

Values

"maxDelta" | "absolute"

When the default value of `maxDelta` is used, the voltage is determined by the maximum delta voltage of two neighboring nets.

When the value is `absolute`, the voltage is determined by the maximum voltage of a net, irrespective of the voltage of the neighboring net.

Parameters

None

widthLengthRatio

```
spacings(  
    ( widthLengthRatio tx_layer  
      'width g_width  
        g_ratio  
    )  
);
```

(Virtuoso Advanced Node for Layout Only) Specifies that the length ratio of shapes with the given width to the total length of all of the shapes on the given layer must be less than or equal to the given ratio.

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ratio</i>	The given ratio, which can be a value in the range 0.0 to 1.0.

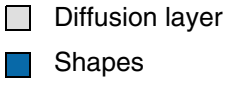
Parameters

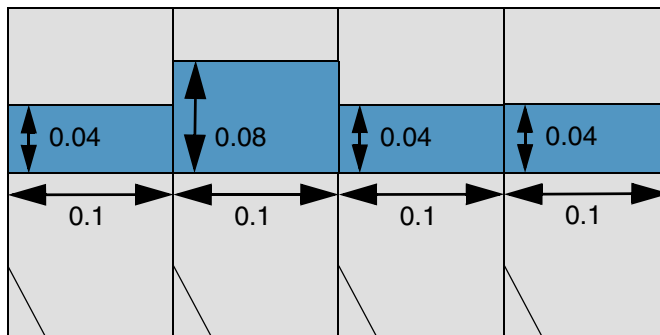
'width <i>g_width</i>	The exact width of the shape to check against the given ratio.

Example

The length ratio of shapes with width 0.08 to the total length of all shapes on the diffusion layer must be less than or equal to the ratio 0.3.

```
.  
spacings (  
    (widthLengthRatio "Diffusion"  
        'width 0.08  
        0.3  
    )  
) ;spacings
```





PASS. The length of shapes with width 0.08 is 0.1 and the total length is 0.4.
The length ratio is $0.1/0.4 = 0.25$ (which is ≤ 0.3).

Virtuoso Technology Data Constraints Reference

Miscellaneous Constraints

NumCut Constraints

This topic includes the following constraints:

- maxViaClusterNumCuts
- minNumCut
- minProtrusionNumCut
- sameMetalAlignedCuts

maxViaClusterNumCuts

```
spacings (
    ( maxViaClusterNumCuts tx_layer
      'cutClass {f_width | (f_width f_length) | t_name}
      'within (f_minWithin f_maxWithin)
      ['diagonalNumCut x_diagonalNumCut
        ['diagonalSpacing f_diagonalSpacing
          'bendSpacing f_bendSpacing
          'diagonalWithin f_diagonalWithin
          'sideExtension f_sideExt
          'edgeExtension f_edgeExt
        ]
      ]
      x_numCuts
    )
) ;spacings
```

Specifies the maximum number of consecutive via cuts of the specified cut class allowed in a cluster and the cut-to-cut spacing that must be satisfied. Via cuts that satisfy the specified spacing are defined to be in the same via cluster.

This constraint allows the spacing between via cuts to be smaller if via cuts are placed in a particular configuration. The via cuts can be placed in a horizontal line, along a straight diagonal line, or diagonally in a bend or notch formation.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_numCuts</i>	The maximum number of via cuts allowed in a cluster.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'within (f_minWithin f_maxWithin)`

The distance between consecutive via cuts in a cluster must be greater than or equal to *minWithin* and less than or equal to *maxWithin*.

Note: For diagonal via cuts (that is, via cuts placed along a straight diagonal line), the distance is measured from the corner of a via cut in both X and Y directions.

`'diagonalNumCut x_diagonalNumCut`

The number of diagonal via cuts in a cluster (that is, via cuts placed along a straight diagonal line) must be less than or equal to this value.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

```
'diagonalSpacing f_diagonalSpacing 'bendSpacing f_bendSpacing  
'diagonalWithin f_diagonalWithin 'sideExtension f_sideExt  
'edgeExtension f_edgeExt
```

These parameters can be specified when *diagonalNumCut* is 0, which implies that there is no limit on the number of via cuts that can be placed diagonally and a bend or notch is allowed.

If via cuts are placed diagonally without a bend or notch, a via cut can have at most two neighboring via cuts at a distance less than or equal to *diagonalWithin*, with spacing greater than or equal to *diagonalSpacing*.

If via cuts are placed to form a bend or notch, a via cut can have at most two non-perfectly-aligned neighboring via cuts in a bend or notch formation at a distance less than or equal to *diagonalWithin*, with spacing greater than or equal to *bendSpacing*.

Additionally, the neighboring via cuts placed along the diagonal or in a bend or notch formation must be found within a search window formed by extending (on both sides) by *sideExt* the via cut edges that satisfy the *minExtensionEdge* constraint with 'twoSides parameter. The other dimension of the search window is equal to *edgeExt*.

All measurements—*diagonalWithin*, *diagonalSpacing*, and *bendSpacing*—are center-to-center.

Examples

- Example 1: maxViaClusterNumCuts with cutClass, within, and diagonalNumCut
- Example 2: maxViaClusterNumCuts with cutClass, within, diagonalNumCut, diagonalSpacing, diagonalWithin, sideExtension, and edgeExtension

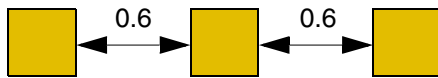
Example 1: maxViaClusterNumCuts with cutClass, within, and diagonalNumCut

The number of 0.3x0.3 Via1 via cuts in a cluster cannot exceed 3. The following additional conditions must also be met:

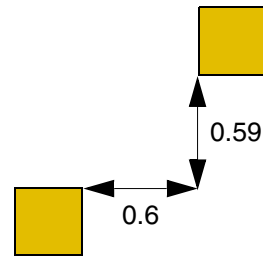
- The distance between via cuts must be greater than or equal to 0.6 and less than or equal to 0.7.
- At most three via cuts can be placed along a straight diagonal line.

```
spacings (
  ( maxViaClusterNumCuts "Via1"
    'cutClass 0.3
    'within (0.6 0.7)
    'diagonalNumCut 3
    3
  )
) ;spacings
```

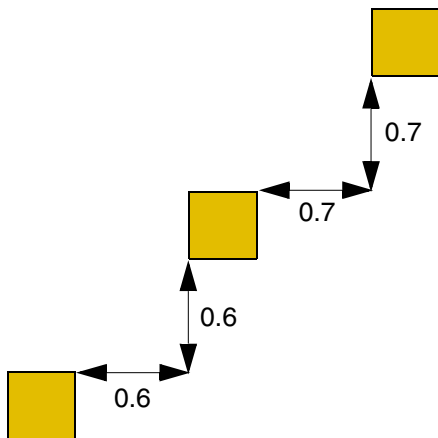
■ Via1



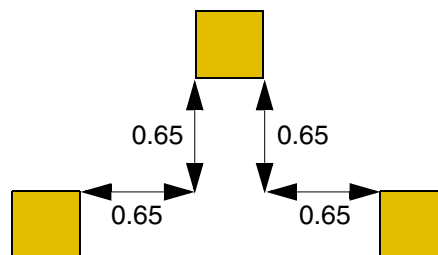
a) PASS. The three via cuts are aligned and are at a distance 0.6 from each other.



b) FAIL. The distance between the via cuts in the Y direction is 0.59 (<0.06).



c) PASS. The three via cuts are placed along a straight diagonal line and the spacing between the via cuts in both X and Y directions is met.



d) FAIL. The via cuts are placed in a bend or notch formation (instead of along a straight diagonal line).

Example 2: maxViaClusterNumCuts with cutClass, within, diagonalNumCut, diagonalSpacing, diagonalWithin, sideExtension, and edgeExtension

The number of 0.3x0.3 Via1 via cuts in a cluster cannot exceed 3. The following additional conditions must also be met:

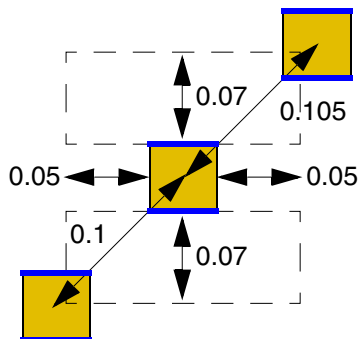
- The via cuts when placed along a straight diagonal line must be within a distance of 0.13 from each other and the spacing between them must be greater than or equal to 0.1.
- The via cuts when placed in a bend or notch formation must be within a distance of 0.13 from each other and the spacing between them must be greater than or equal to 0.11.
- The via cuts that satisfy the two conditions listed above must be found within a search window formed by extending by 0.05 the via cut edges that satisfy the minExtensionEdge constraint with 'twoSides parameter (in blue) on both sides. The other dimension of the search window is equal to 0.07.

```

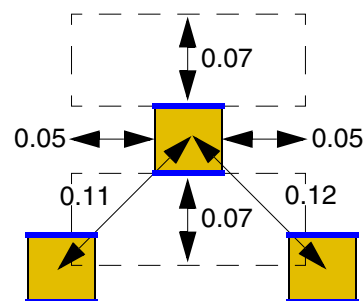
spacings (
  ( maxViaClusterNumCuts "Via1"
    'cutClass 0.3
    'within (0.6 0.7)
    'diagonalNumCut 0
    'diagonalSpacing 0.1
    'bendSpacing 0.11
    'diagonalWithin 0.13
    'sideExtension 0.05
    'edgeExtension 0.07
    3
  )
) ;spacings

```

■ Via1
□ Search window



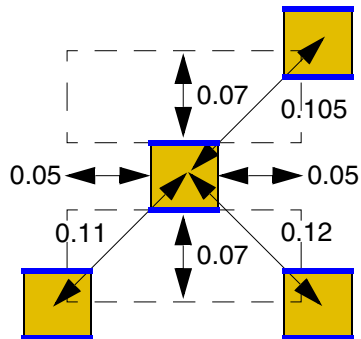
a) PASS. The via cuts are placed along a straight diagonal line within a distance of 0.13 from each other. Additionally, the via cuts overlap the search window and the spacing between the via cuts is greater than or equal to 0.1.



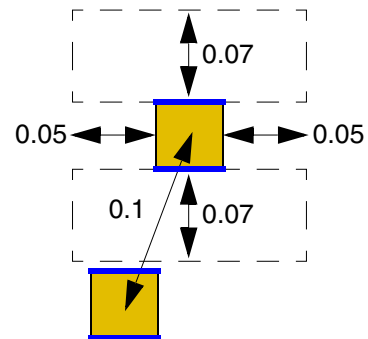
b) PASS. The via cuts are placed along a bend or notch within a distance of 0.13 from each other. Additionally, the via cuts overlap the search window and the spacing between the via cuts is greater than or equal to 0.11.

Virtuoso Technology Data Constraints Reference

NumCut Constraints



c) The constraint does not apply because the number of neighboring vias within a distance of 0.13 can be at most 3.

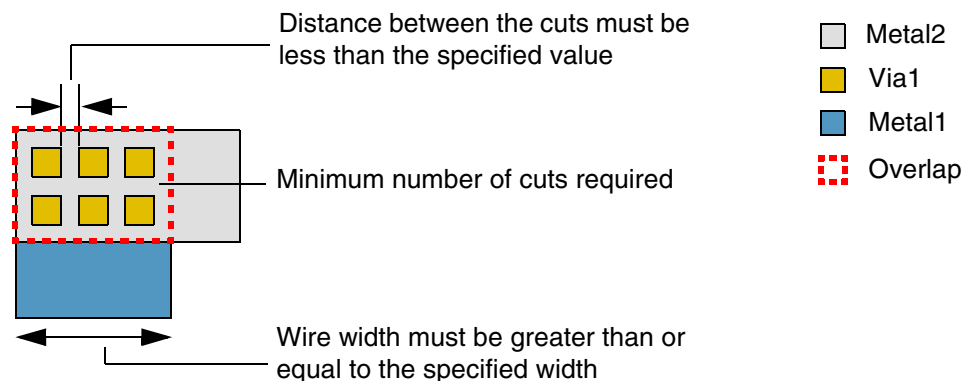


d) FAIL. The neighboring via cut has parallel run length greater than zero and is outside the search window. This implies that the two via cuts are not aligned diagonally.

minNumCut

```
spacingTables (
  ( minNumCut tx_cutLayer
    (( "width" nil nil )
      ['cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
      ['diffCutClass]]
      ['distanceWithin f_within ['manhattan]]
      ['above | 'below | 'bothAboveBelow]
      ['sameMetalOverlap | 'fullyEnclosed]
      [f_default]
      ['otherWidth g_otherWidth]
    )
    (g_table)
  )
) ;spacingTables
```

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. If 'distanceWithin is specified, only the via cuts that are within the specified distance from each other are counted.



Usually, wide wires carry more current, and if you change layers between two wide wires, you need a sufficient number of via cuts to carry an equivalent amount of current. Using multiple via cuts, instead of just one, also increases reliability.

The `minNumCut` constraint can be specified in the following constraint groups:

- **Wire Edit:** The `minNumCut` constraint can be specified in a wire edit constraint group, which can then be set as the default constraint group for creating and editing wires by using either the `wireConstraintGroup` environment variable or by selecting it in the *Options – Editor – Default Wire Constraint Group* list.
- **Foundry:** The definition of `minNumCut` in the wire edit constraint group overrides the `minNumCut` definition available in the foundry constraint group.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

- **Net:** The `minNumCut` constraint can be specified for a particular net in a net constraint group. Currently, the router supports only a limited number of constraints in the net constraint group, including `minWidth`, `minNumCut`, `minSpacing`, `validLayers`, and `validVias`.

Values

<code>tx_cutLayer</code>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>"width" nil nil</code>	This identifies the index for <code>table</code> .
<code>g_table</code>	The format of the table is as follows: (<code>f_width</code> <code>x_numberOfCuts</code>) where, ■ <code>f_width</code> is the width of the wire. The width of the wire must be greater than or equal to this value for the number of corresponding via cuts to apply. ■ <code>x_numberOfCuts</code> is the minimum number of cuts that a via must contain. Type: A 1-D table specifying floating-point width values and the number of via cuts required corresponding to different widths

Parameters

<code>'cutClass {f_width (f_width f_length) t_name}</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <code>f_width</code>: Width ■ <code>f_length</code>: Length ■ <code>t_name</code>: Name of the cut class

Virtuoso Technology Data Constraints Reference

NumCut Constraints

<code>'diffCutClass</code>	(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that if there is one cut belonging to the cut class in a wide wire, there must be at least the specified cut number minus 1 cuts not belonging to the cut class.
<code>'distanceWithin</code> <i>f_within</i>	The distance between the via cuts must be less than this value for the via cuts to be counted.
<code>'distanceMeasureType</code>	(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the within distance should be measured as a euclidian or manhattan. The default is euclidian.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

'above | 'below | 'bothAboveBelow

The wire that is used to determine the number of cuts required. By default, the number of cuts required is determined based on the width of the wider of the two wires.

- 'above: The number of cuts required is determined based on the width of the wire on the layer above.
- 'below: The number of cuts required is determined based on the width of the wire on the layer below.
- 'bothAboveBelow: (Virtuoso Advanced Node for Layout Standard) The number of cuts required is determined based on the width of the narrower of the two wires.

'sameMetalOverlap | 'fullyEnclosed

The minimum number of cuts required is determined based on the width of the wider wire.

- 'sameMetalOverlap: (Virtuoso Advanced Node for Layout Standard) The constraint applies only if the overlap between the wires on the layers above and below the cut layer contains the via cuts and partly covers the wider wire. The via cuts need not overlap the wider wire.
- 'fullyEnclosed: (Virtuoso Advanced Node for Layout Only) All via cuts must be completely inside a wide wire to be counted against the minimum via cut requirement.

Type: Boolean

f_default

The number of cuts to be used when no table entry applies.

'otherWidth *g_otherWidth*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the wire width on the other layer is less than the specified value.

Examples

- [Example 1: minNumCut with above](#)
- [Example 2: minNumCut with bothAboveBelow](#)
- [Example 3: minNumCut with sameMetalOverlap](#)

Virtuoso Technology Data Constraints Reference

NumCut Constraints

- Example 4: minNumCut with fullyEnclosed

Virtuoso Technology Data Constraints Reference

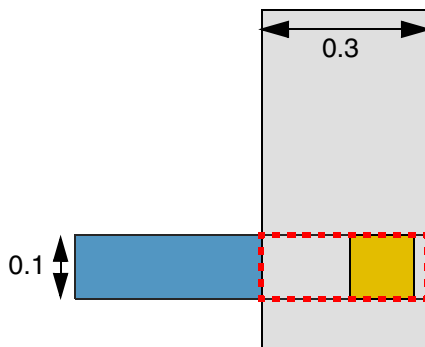
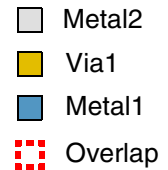
NumCut Constraints

Example 1: minNumCut with above

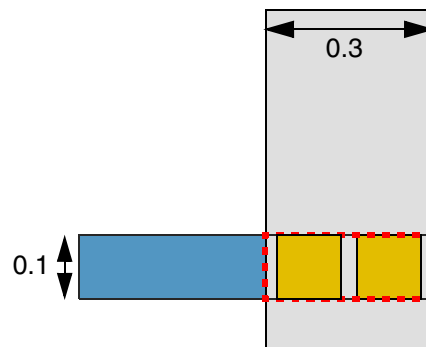
The width of the wire on the layer above determines the number of cuts that a via must contain.

- Width ≥ 0.1 ; via cuts = 1
- Width ≥ 0.3 ; via cuts = 2
- Width ≥ 0.5 ; via cuts = 4
- Width ≥ 0.6 ; via cuts = 6

```
spacingTables(  
  ( minNumCut "Via1"  
    (( "width" nil nil )  
      'above  
    )  
    (  
      0.1 1  
      0.3 2  
      0.5 4  
      0.6 6  
    )  
  )  
) ;spacingTables
```



a) FAIL. The width of the wire on the layer above is 0.3. Therefore, at least two via cuts are required.



b) PASS. The width of the wire on the layer above is 0.3 and the number of via cuts is 2.

Virtuoso Technology Data Constraints Reference

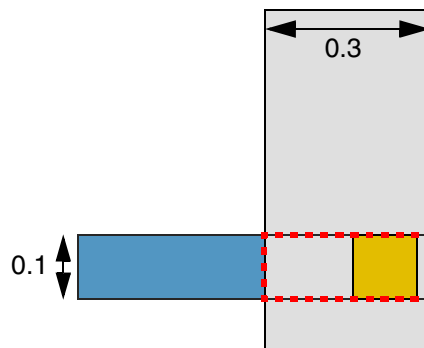
NumCut Constraints

Example 2: *minNumCut with bothAboveBelow*

The width of the wires on the layers above and below determine the number of cuts that a via must contain.

- Width ≥ 0.1 ; via cuts = 1
- Width ≥ 0.3 ; via cuts = 2
- Width ≥ 0.5 ; via cuts = 4
- Width ≥ 0.6 ; via cuts = 6

```
spacingTables (
  ( minNumCut "Via1"
    (( "width" nil nil )
      'bothAboveBelow
    )
    (
      0.1 1
      0.3 2
      0.5 4
      0.6 6
    )
  )
) ;spacingTables
```



PASS. The narrower of the two wires is 0.1 wide. Therefore, the minimum number of via cuts required is 1.

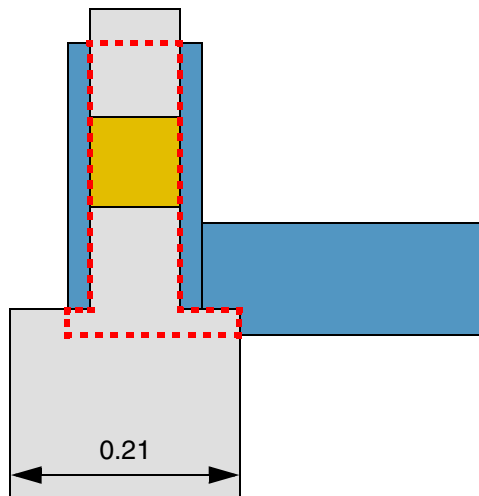
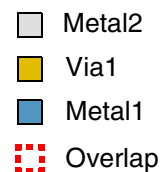
Example 3: *minNumCut* with *sameMetalOverlap*

The width of the wider wire determines the number of cuts that a via must contain.

- Width ≥ 0.0 ; via cuts = 1
- Width ≥ 0.2 ; via cuts = 2

The via is contained in the overlap of the two wires, and this overlap partly covers the wider of the two wires.

```
spacingTables(  
  ( minNumCut "Via1"  
    (( "width" nil nil )  
      'sameMetalOverlap  
    )  
    (  
      0.0 1  
      0.2 2  
    )  
  )  
) ;spacingTables
```



FAIL. The constraint applies because the via is contained in the overlap of the two wires and this overlap partly covers the wider of the two wires. However, the via contains just one cut (the number of cuts required by a wire with width 0.21 is two).

Example 4: minNumCut with fullyEnclosed

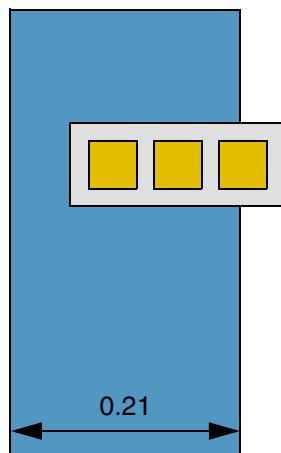
The width of the wider wire determines the number of cuts that a via must contain.

- Width ≥ 0.0 ; via cuts = 1
- Width ≥ 0.1 ; via cuts = 2
- Width ≥ 0.2 ; via cuts = 3

All via cuts must be fully enclosed by the wide wire.

```
spacingTables(  
  ( minNumCut "Via1"  
    (( "width" nil nil )  
      'fullyEnclosed  
    )  
    (  
      0.0 1  
      0.1 2  
      0.2 3  
    )  
  )  
) ;spacingTables
```

□ Metal2
■ Via1
■ Metal1



FAIL. Only two via cuts (instead of the required three) are fully enclosed by the wide wire with width 0.21. The constraint would be satisfied if 'fullyEnclosed' is not specified.

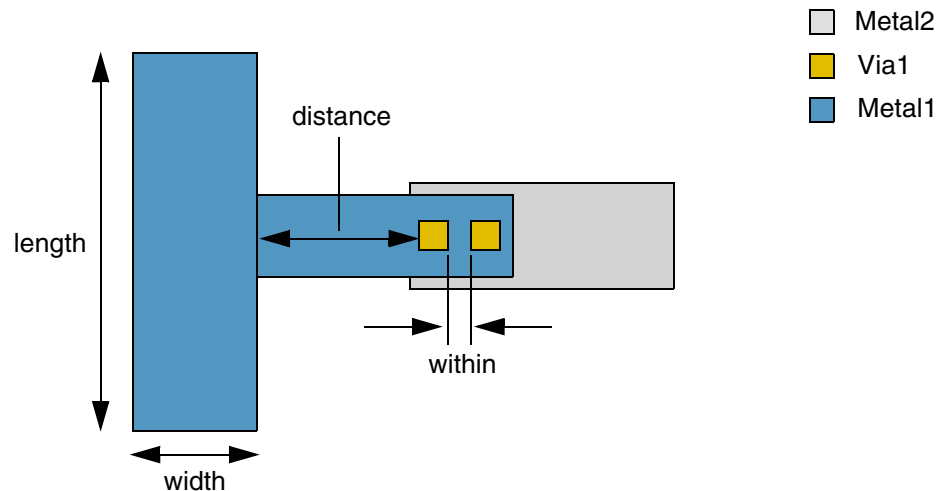
minProtrusionNumCut

```

spacings (
  ( minProtrusionNumCut tx_cutLayer
    ['cutClass {f_width | (f_width f_length) | t_name}]
    ['distance f_distance]
    'width f_width
    {'area f_area ['distance f_distance] | 'length f_length
      'distance f_distance }
    ['distanceWithin f_within]
    ['above | 'below]
    x_numCuts
  )
  ( minProtrusionNumCut tx_cutLayer
    (f_distance f_length f_width x_numCuts)
  )
) ;spacings

```

Specifies the minimum number of cuts a via must have when the via is placed on a thin wire (protrusion) directly connected to a wide wire or pin.



Virtuoso Technology Data Constraints Reference

NumCut Constraints

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_numCuts</i>	The number of via cuts must be greater than or equal to this value.

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

`'distance f_distance`

The constraint applies only if the distance of the first via cut (on the protrusion) from the wide wire is less than this value.

`'width f_width`

The constraint applies only if the width of the wide wire is greater than this value.

`'length f_length`

The constraint applies only if the length of the wide wire is greater than this value.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

'area *f_area*

The constraint applies only if the width of the wide wire is greater than *width* and the area of the island comprising the wide wire and the protrusion is greater than this value.

If *width* is less than the width of the default routing wire and is specified along with *area*, the minimum number of cuts required on the routing vias varies based on the area of the routing wire on the layer.

If only *area* and *width* are specified, only the thin wire connected to a wide wire is checked for the required number of via cuts.

'distanceWithin *f_within*

The distance between the via cuts must be less than this value for the via cuts to be counted.

'above | 'below

The metal layer to which the constraint applies. By default, the constraint applies to wires on the metal layers directly above and below the cut layer.

- 'above: The constraint applies to the wires on the layer above.
- 'below: The constraint applies to the wires on the layer below.

Examples

- Example 1: minProtrusionNumCut with width, area, and distanceWithin
- Example 2: minProtrusionNumCut with distance, width, area, length, and distanceWithin
- Example 3: minProtrusionNumCut with distance, length, width

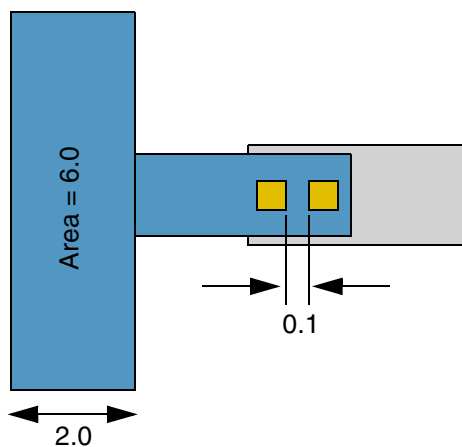
Example 1: minProtrusionNumCut with width, area, and distanceWithin

The minimum number of via cuts required on a protrusion from a wide wire is 2 if the following conditions are met:

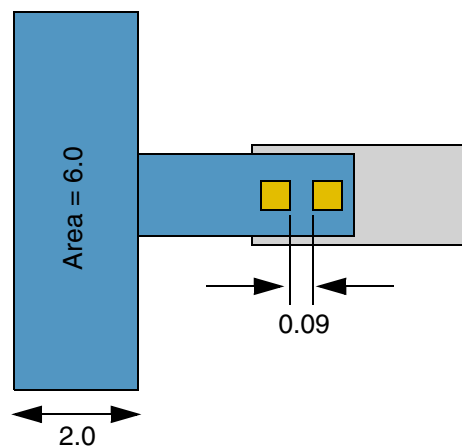
- The width of the wide wire is greater than 1.0.
- The area of the island comprising the wide wire and the protrusion is greater than 6.0.
- The distance between via cuts is less than 0.1.

```
spacings(
  ( minProtrusionNumCut "Via1"
    'width 1.0
    'area 6.0
    'distanceWithin 0.1
    2
  )
); spacings
```

Metal2
 Via1
 Metal1



a) FAIL. The area of the wide wire is greater than 6.0 and its width is greater than 1.0. Additionally, the number of via cuts is 2. However, the distance between the via cuts is 0.1 (and not less than 1.0).



b) PASS. The area of the wide wire is greater than 6.0 and its width is greater than 1.0. Additionally, the number of via cuts is 2 and the distance between the via cuts is 0.09 (< 1.0).

Example 2: minProtrusionNumCut with distance, width, area, length, and distanceWithin

The minimum number of via cuts required on a protrusion from a wide wire is 2 if all of the following conditions are met:

- The distance of the first via cut (on the protrusion) from the wide wire is less than 0.5.
- The width of the wide wire is greater than 2.0.
- The length of the wide wire is greater than 3.0.

If all of the conditions listed above are met and the distance between via cuts is less than 0.3, the minimum number of via cuts required is also 2.

```
spacings (
  ( minProtrusionNumCut "cut1"
    'distance 0.5
    'width 2.0
    'length 3.0
    2
  )
  ( minProtrusionNumCut "cut2"
    'distance 0.5
    'width 2.0
    'length 3.0
    'distanceWithin 0.3
    2
  )
) ;spacings
```

Example 3: minProtrusionNumCut with distance, length, width

This example shows an alternative way to use the minProtrusionNumCut constraint. The distance, length, and width are 0.1, 0.2, and 04, respectively, and the minimum number of via cuts is 4.

```
spacings (
  ( minProtrusionNumCut "cut1"
    (0.1 0.2 0.4 4)
  )
) ;spacings
```

sameMetalAlignedCuts

```
orderedSpacings (
    ( sameMetalAlignedCuts tx_cut tx_metal
        ['cutClass {g_width | (g_width g_length) | t_name}]
        'width g_width
        'distance g_cutDistance
        ['prl g_parallelRunLength 'within g_within
            | 'parallelEdgeWithin g_metalWithin ['withCut]
            | 'cutWithin g_cutWithin
        ]
        ['cutClassSetList g_cutClassSetList]
        ['noViaMetalOverlap g_noViaMetalOverlap
            'otherLayer g_otherLayer]
        ['sameViaConnectivity]
        ['horizontal | 'vertical]
        ['centerToCenterSpacing | 'edgeToEdgeSpacing]
        ['otherMask1 | 'otherMask2 | 'otherMask3 | 'otherMask4]
        g_numCuts
    )
); orderedSpacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the maximum number of centered cuts if the cuts share the same shape above or below the metal layer.

Values

<i>tx_cut</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_metal</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_numCuts</i>	The maximum number of cut shapes.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

Parameters

`'cutClass {g_width | (g_width g_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `g_width`: Width
- `g_length`: Length
- `t_name`: Name of the cut class. If not specified, the constraint applies to all cut classes.

`'width g_width`

The constraint applies only if the width of the metal containing the cuts is less than or equal to this value.

`'distance g_cutDistance`

The constraint applies only if the spacing between the cuts is less than this value. The `g_cutDistance` value is measured center-to-center or edge-to-edge depending on other parameters selected.

`'prl f_parallelRunLength`

The constraint applies only if the layer with same-metal cuts has a neighboring wire within `f_within` and a parallel run length greater than `f_parallelRunLength` to the cuts. The width of the neighboring wire must be less than or equal to `g_width`.

`'within f_within`

The constraint applies only if the distance of the neighboring wire is within `f_within`.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

`'parallelEdgeWithin g_metalWithin ['withCut]`

The constraint applies only if there are neighbor wires on one or both sides on the metal layer, which all the cuts share, within the given value and touching or overlapping with more than *g_numCuts*. Neighbor wires on opposite sides would be merged only if they overlap. In addition, the width of this neighbor wire must also be less than or equal to *g_width*. The *g_cutDistance* value is measured edge-to-edge.

The *withCut* parameter specifies that the constraint applies only if a neighboring wire contains a cut on the adjacent cut layer, which touches or overlaps with a window formed by the violating cuts.

`'cutWithin g_cutWithin`

The constraint applies only if a neighbor wire contains a cut on the adjacent cut layer of the current layer with parallel run length more than 0 and with a window formed by merging the violating cuts. The neighbor wire does not need to cover all the aligned cuts. The *g_cutDistance* value is measured edge-to-edge.

`'cutClassSetList g_cutClassSetList`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if all cuts belong to cut classes in the list and if there is at least one cut from each cut class set. For example, if the value is ("VA" "VB" " ", " " "VA" "VC"), at least one cut must be VA, at least one cut must be VB or VC, and other cuts can be of any cut class in the list.

`'noViaMetalOverlap g_noViaMetalOverlap 'otherLayer g_otherLayer`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if there is a shape on *g_otherLayer* overlapping with the metal covering the aligned cuts by extending the given *g_noViaMetalOverlap* value on one of the end cuts of the aligned cuts and there is no via cut on the overlap metal.

`'sameViaConnectivity`

The constraint applies only if the via cuts are overlapped by a single metal shape from above and by another single metal shape from below.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

'horizontal | 'vertical

The constraint applies only to the cut spacing in the specified direction.

'centerToCenterSpacing | 'edgeToEdgeSpacing

The cut spacing is measured as center-to-center, which is the default, or edge-to-edge.

'otherMask1 | 'otherMask2 | 'otherMask3 | 'otherMask4

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies to the metal mask containing the aligned cuts, otherMask1, otherMask2, otherMask3, or otherMask4.

Examples

- [Example 1: sameMetalAlignedCuts with width and distance](#)
- [Example 2: sameMetalAlignedCuts with prl and within](#)
- [Example 3: sameMetalAlignedCuts with parallelEdgeWithin](#)
- [Example 4: sameMetalAlignedCuts with cutWithin](#)
- [Example 5: sameMetalAlignedCuts with noViaMetalOverlap](#)
- [Example 6: sameMetalAlignedCuts with sameViaConnectivity](#)

Virtuoso Technology Data Constraints Reference

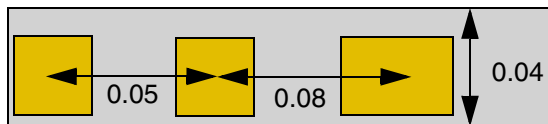
NumCut Constraints

Example 1: *sameMetalAlignedCuts with width and distance*

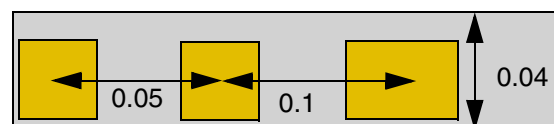
The maximum number of centered cuts is 2 when the cuts share the same metal above or below the metal layer and when the following conditions are met.

```
orderedSpacings(  
  ( sameMetalAlignedCuts "Via1" "Metal1"  
    'width 0.04  
    'distance 0.1  
    2  
  )  
) ;orderedSpacings
```

□ Metal1
■ Via1



a) FAIL. There are 3 aligned vias (more than the specified number) with a distance of less than 0.1.



b) PASS. There are 2 aligned vias with distance < 0.1. The right via is not an aligned via because its distance is not < 0.1 from the neighbor.

Example 2: *sameMetalAlignedCuts with prl and within*

The maximum number of centered cuts with center-to-center spacing less than 0.1 is 2 when these conditions are met:

- The cuts share the same metal above or below the metal layer
- There is a shape on layer tx_metal within 0.03μm with parallel run length more than 0.05μm

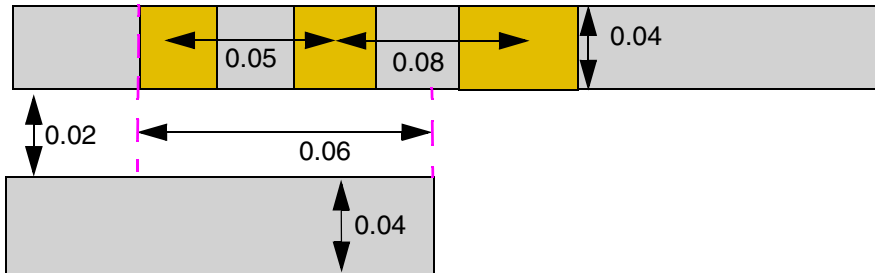
Virtuoso Technology Data Constraints Reference

NumCut Constraints

- Both tx_metal shapes have width less than or equal to 0.04

```
orderedSpacings (
  ( sameMetalAlignedCuts "Via1" "Metal1" 'cutClass "VA"
    'width 0.04 'distance 0.1
    'prl 0.05 'within 0.03
    2
  )
) ;orderedSpacings
```

Metal1
 Via1



FAIL. The number of center align cuts of any cut classes sharing the same metal on the lower metal layer with center-to-center spacing < 0.1 is 3 (> 2). There is a below metal layer within 0.03 and with parallel run length of 0.06 (> 0.05) to the cuts.

Example 3: sameMetalAlignedCuts with parallelEdgeWithin

The maximum number of centered cuts with edge-to-edge spacing less than 0.1 is 2 when these conditions are met:

- The cuts share the same metal above or below the metal layer
- Neighbor wires on one or both sides on the metal layer that all the cuts share are within $0.03\mu\text{m}$.

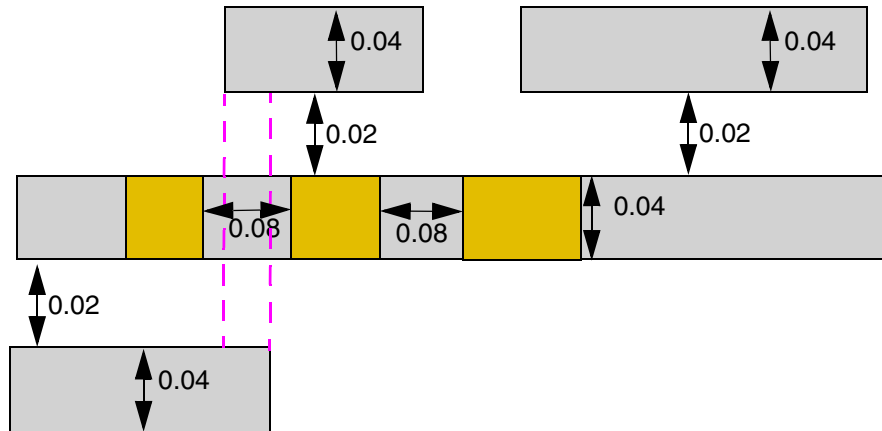
Virtuoso Technology Data Constraints Reference

NumCut Constraints

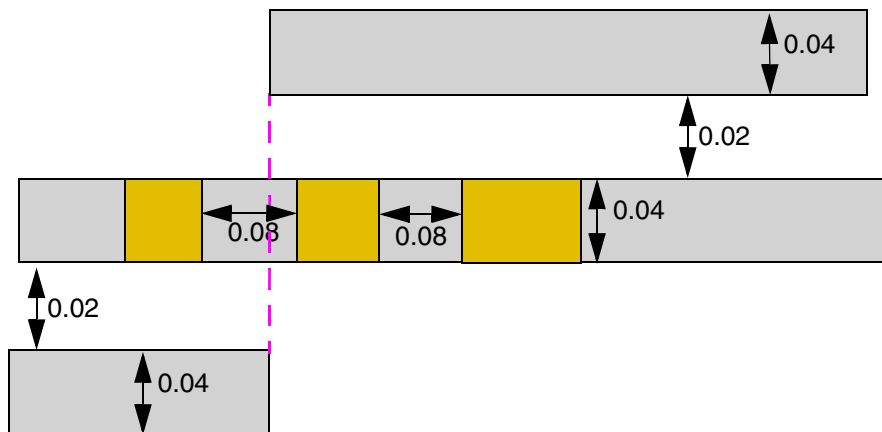
- Both tx_metal shapes have width less than or equal to 0.04

```
orderedSpacings (
  ( sameMetalAlignedCuts "Via1" "Metal1" 'cutClass "VA"
    'width 0.04 'distance 0.1 'parallelEdgeWithin 0.03
    2
  )
) ;orderedSpacings
```

Metal1
 Via1



- a. Pass. The left two wires combine to overlap with two cuts and the right wire overlaps only one cut.



- a. Fail. The two neighbor wires combine to overlap with 3 (more than 2) aligned cuts.

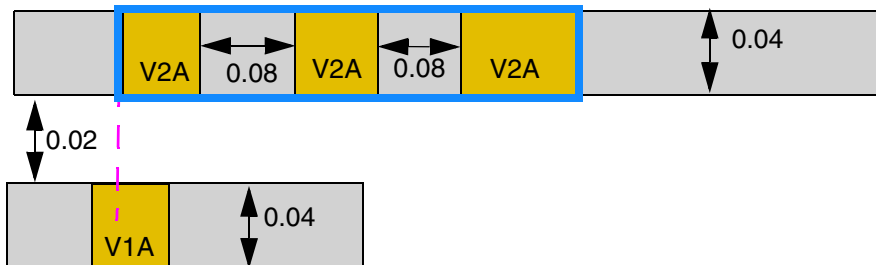
Example 4: *sameMetalAlignedCuts with cutWithin*

The maximum number of centered cuts with edge-to-edge spacing less than 0.1 is 2 when these conditions are met:

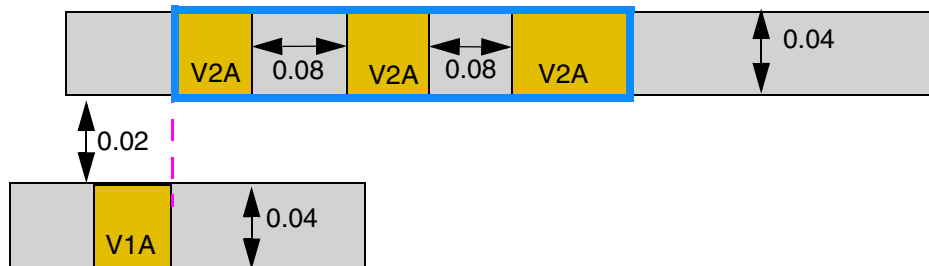
- The cuts share the same metal above or below the metal layer
- A neighbor wire within $0.03\mu\text{m}$ contains a cut on the adjacent cut layer of the current layer
- Both tx_metal shapes have width less than or equal to 0.04

```
orderedSpacings (
  ( sameMetalAlignedCuts "Vial" "Metal1" 'cutClass "VA"
    'width 0.04 'distance 0.1 'cutWithin 0.03
    2
  )
) ;orderedSpacings
```

□ Metal1
■ Via1



a) Fail. There is an adjacent layer cut, V1A, having PRL more than 0 with the blue window formed by merging the violating cuts of V2A, and the neighbor wire does not cover all the aligned cuts.



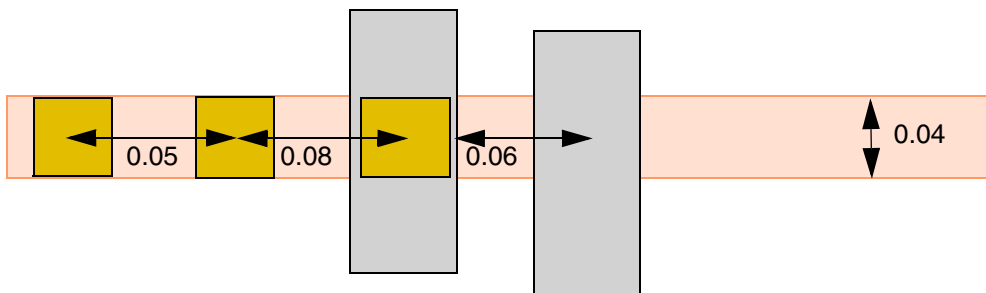
b) Pass. No adjacent layer cut is found having PRL more than 0 with the blue window because V1A is just outside the window.

Example 5: *sameMetalAlignedCuts with noViaMetalOverlap*

The maximum number of centered cuts is 2 when the cuts share the same metal above or below the metal layer and when a shape on Metal2 overlaps with the metal covering the aligned cuts by extending by 0.06.

```
orderedSpacings (
  ( sameMetalAlignedCuts "Via1" "Metal1"
    'cutClass "VA"
    'width 0.04
    'distance 0.1
    'noViaMetalOverlap 0.06
    'otherLayer "Metal2"
    2
  )
) ;orderedSpacings
```

Metal1
 Metal2
 Via1



FAIL. An above metal within 0.06 exists on the above metal wire connected to the right cut and that metal overlaps with the below metal without a via cut.

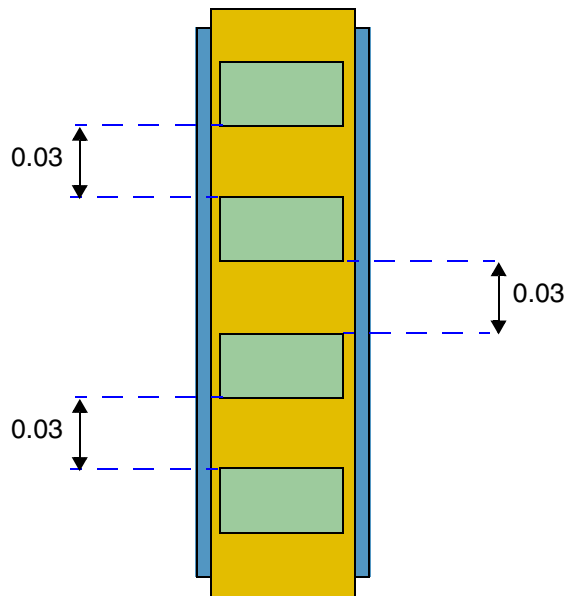
Example 6: *sameMetalAlignedCuts with sameViaConnectivity*

The maximum number of centered cuts is 2 when these conditions are met:

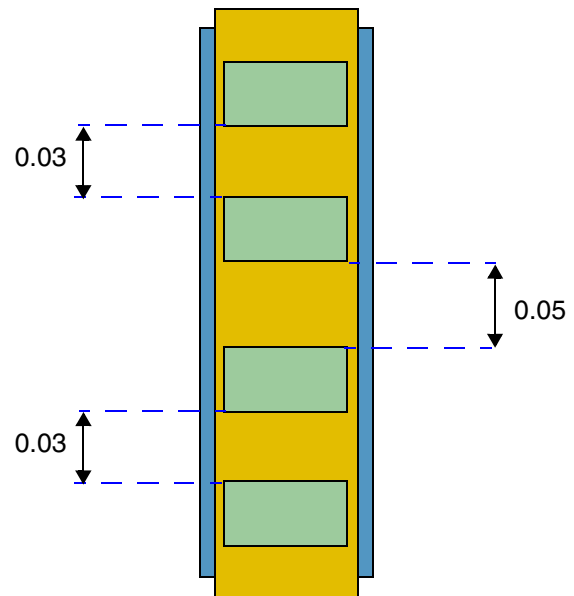
- The cuts share the same metal above and below the metal layer
- Both *tx_metal* shapes have width less than or equal to 0.06.
- The via cuts are overlapped by a single metal shape from above and by another single metal shape from below.
- The edge-to-edge spacing in vertical direction is less than 0.05.

```
orderedSpacings (
  ( sameMetalAlignedCuts "Via1" "Metal1"
    'cutClass "VA"
    'width 0.06
    'distance 0.05
    'sameViaConnectivity
    'vertical
    'edgeToEdgeSpacing
    2
  )
) ;orderedSpacings
```

Metal2
 Metal1
 Via1



The constraint fails because the distance is less than 0.05.



The constraint passes because the distance is not less than 0.05.

Virtuoso Technology Data Constraints Reference

NumCut Constraints

Overlap Constraints

This topic includes the following constraints:

- minDirectionalOverlap
- minInsideCornerOverlap
- minOverlapDistance
- minWireOverlap
- maxOverlap
- overlapNotAllowed

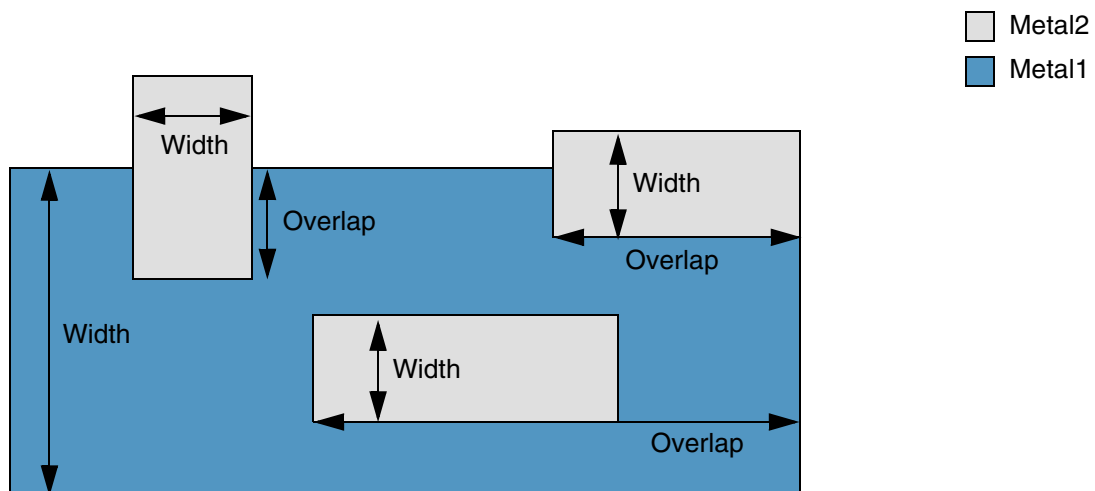
minDirectionalOverlap

```
orderedSpacings (
    ( minDirectionalOverlap tx_layer1 tx_layer2
      ['horizontalOverlap | 'verticalOverlap]
      ['widthRanges g_ranges]
      ['otherWidthRanges g_ranges]
      ['sameNetOnly]
      f_value
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum overlap required between the shapes on *layer1* and *layer2*. The overlap is measured in the direction of the length of the *layer2* shape, between the outside edge of the *layer1* shape and the innermost edge of the *layer2* shape.

Optionally, the constraint applies when one or more of the following conditions are met:

- The *layer1* shape is within the specified width range.
- The *layer2* shape is within the specified width range.
- The overlap is measured in the horizontal or vertical direction.
- The *layer1* and *layer2* shapes are on the same net.



Virtuoso Technology Data Constraints Reference

Overlap Constraints

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_value</i>	The overlap must be greater than or equal to this value.

Parameters

<code>'horizontalOverlap 'verticalOverlap</code>	The direction in which the overlap is measured. By default, the constraint applies to both vertical and horizontal overlaps. Type: Boolean
<code>'widthRanges g_ranges</code>	The constraint applies only if the width of the <i>layer1</i> shape falls in this range. Type: Floating-point values specifying a <u>range</u> of widths that are allowed
<code>'otherWidthRanges g_ranges</code>	The constraint applies only if the width of the <i>layer2</i> shape falls in this range. Type: Floating-point values specifying a <u>range</u> of widths that are allowed
<code>'sameNetOnly</code>	The constraint applies only if the shapes on <i>layer1</i> and <i>layer2</i> are on the same net. Type: Boolean

Virtuoso Technology Data Constraints Reference

Overlap Constraints

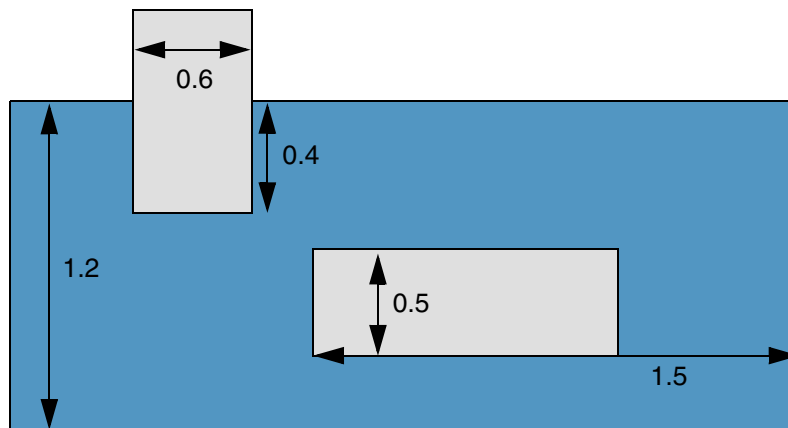
Example

The minimum overlap between Metal1 and Metal2 shapes must be 0.5 if both of the following conditions are met:

- The width of Metal1 shapes is greater than or equal to 1.1 and less than or equal to 1.3.
- The width of Metal2 shapes is greater than or equal to 0.5 and less than or equal to 0.6.

```
orderedSpacings (  
  ( minDirectionalOverlap "Metal1" "Metal2"  
    'widthRanges [1.1 1.3]  
    'otherWidthRanges [0.5 0.6]  
    0.5  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



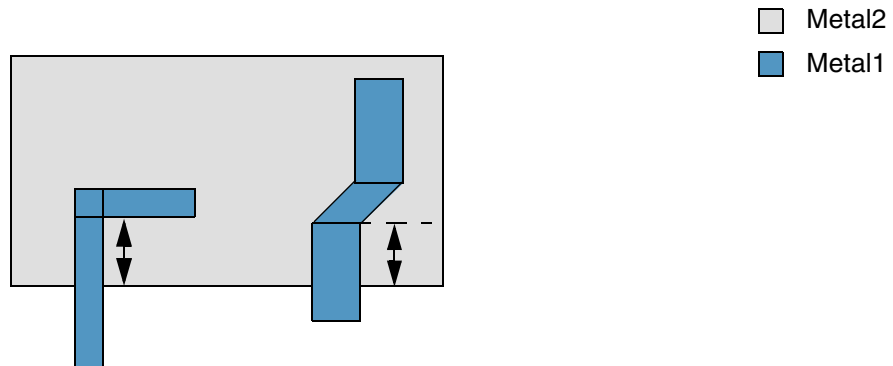
FAIL. The widths of the Metal1 and Metal2 shapes fall in the specified ranges. However, the overlap between the top Metal2 shape and the Metal1 shape is only 0.4 (<0.5).

minInsideCornerOverlap

```
orderedSpacings (
    ( minInsideCornerOverlap tx_layer1 tx_layer2
      f_space
    )
) ;orderedSpacings
```

Specifies the minimum overlap of a shape on *layer2* over an inside corner of a shape on *layer1*.

The constraint applies to shapes with both 45 and 90 degree bends.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_space</i>	The overlap must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Overlap Constraints

Example

The overlap of a Metal2 shape on an inside corner of a Metal1 shape must be at least 1.0.

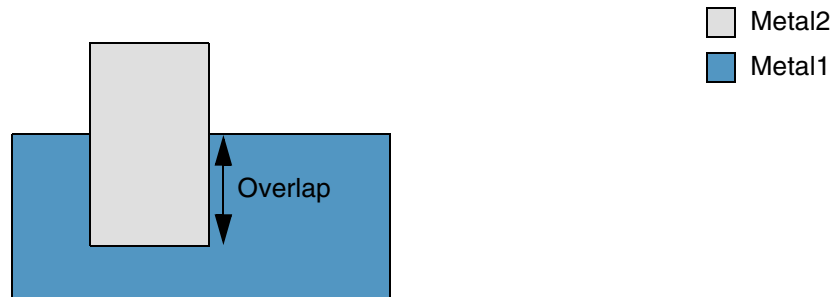
```
orderedSpacings (
  ( minInsideCornerOverlap "Metal1" "Metal2"
    1.0
  )
) ;orderedSpacings
```

minOverlapDistance

```
orderedSpacings (  
    ( minOverlapDistance tx_layer1 tx_layer2  
      f_overlap  
    )  
) ;orderedSpacings
```

Specifies the minimum distance a shape on one layer must overlap a shape on another layer. The overlap distance is measured between the inside edges of the two shapes.

The order of the layers does not matter because the same minimum overlap applies irrespective of whether *layer1* overlaps *layer2* or *layer2* overlaps *layer1*.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_overlap</i>	The overlap between the shapes must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Overlap Constraints

Example

The overlap distance for Metal1 and Metal2 shapes must be at least 2.5.

```
orderedSpacings (
  ( minOverlapDistance "Metal1" "Metal2"
    2.5
  )
) ;orderedSpacings
```

minWireOverlap

```
orderedSpacings(
    ( minWireOverlap tx_layer1 tx_layer2
      ['sameDir] ['centerLine | 'sameWidth]
      f_overlap
    )
) ;orderedSpacings

spacingTables(
    ( minWireOverlap tx_layer1 tx_layer2
      (( "width" nil nil )
        ['sameDir] ['centerLine | 'sameWidth]
        [f_default]
      )
      (g_table)
    )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum overlap of a wire on *layer1* and a wire on *layer2* in the direction of length of the two wires.

The constraint applies only if the wires overlap in the same direction. Optionally, the constraint can restrict any overlap between the wires to be in the same direction.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_overlap</i>	The overlap between the two wires must be greater than or equal to this value.
"width" nil nil	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Overlap Constraints

g_table

The format of the *table* row is as follows:

(f_width f_overlap)

where, *width* is the width of the wire and *overlap* is the minimum overlap requirement that must be satisfied when the width of the wire is greater than or equal to the specified index value.

Type: A 1-D table specifying floating-point width and overlap values

Parameters

'sameDir

The wires can overlap only if they are in the same direction.

Type: Boolean

'centerLine | 'sameWidth

The wire alignment that must be satisfied.

- 'centerLine: The centerlines of the two wires must overlap exactly.
- 'sameWidth: The two wires must have the same width and must completely overlap.

Type: Boolean

f_default

The overlap value to be used when no table entry applies.

Examples

- Example 1: minWireOverlap with sameDir and centerLine
- Example 2: minWireOverlap with sameWidth

Virtuoso Technology Data Constraints Reference

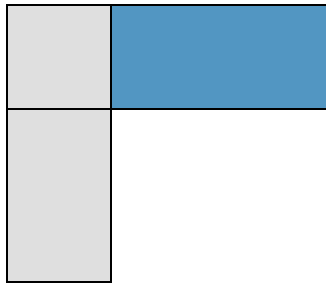
Overlap Constraints

Example 1: minWireOverlap with sameDir and centerLine

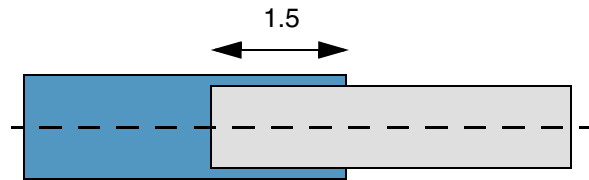
Wires on Metal1 and Metal2 can overlap only if they are in the same direction. Additionally, they must satisfy a minimum overlap value of 1.5 and must be aligned on their centerlines.

```
orderedSpacings (
  ( minWireOverlap "Metal1" "Metal2"
    'sameDir 'centerLine
    1.5
  )
) ;orderedSpacings
```

□ Metal2
■ Metal1



a) FAIL. Only wires in the same direction can overlap.



b) PASS. Wires that overlap are in the same direction and are aligned on the centerline. Additionally, the overlap value is 1.5.

Virtuoso Technology Data Constraints Reference

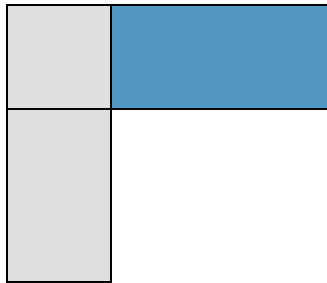
Overlap Constraints

Example 2: *minWireOverlap* with *sameWidth*

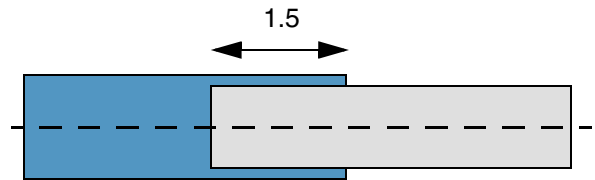
Wires on Metal1 and Metal2 must satisfy a minimum overlap value of 1.5. Additionally, they must have the same width and overlap exactly.

```
orderedSpacings (
  ( minWireOverlap "Metal1" "Metal2"
    'sameWidth
    1.5
  )
) ;orderedSpacings
```

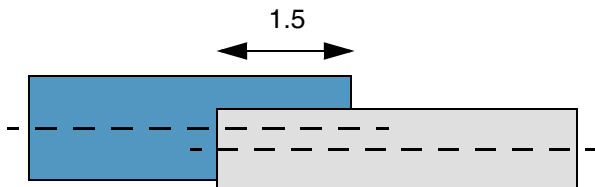
□ Metal2
■ Metal1



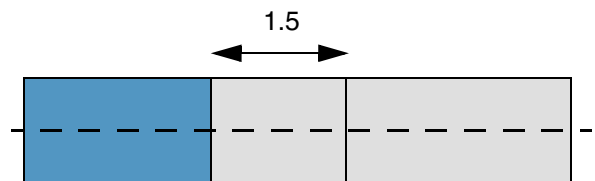
a) The constraint does not apply because the wires are perpendicular to each other. A violation would occur if 'sameDir' was specified.



b) FAIL. The wires do not have the same width.



c) FAIL. Wires must completely overlap exactly.

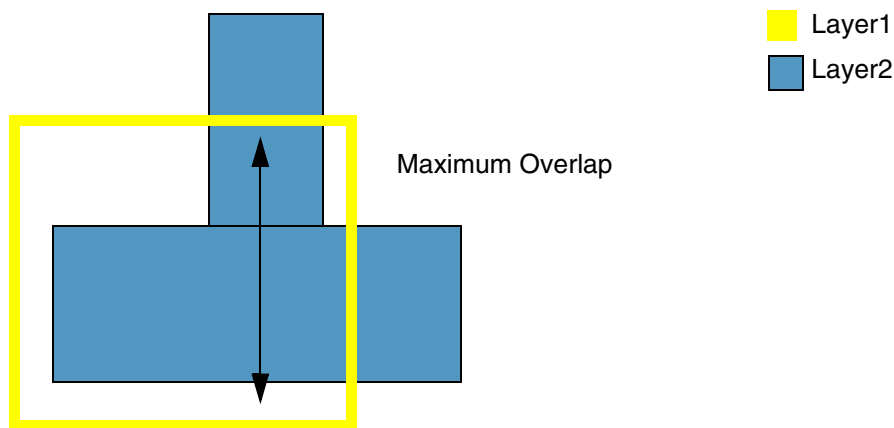


d) PASS. Wires have the same width and the overlap value is 1.5.

maxOverlap

```
orderedSpacings (
    ( maxOverlap tx_layer1 tx_layer2
      g_length
    )
) ;orderedSpacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the maximum distance a shape on one layer can overlap a shape on another layer.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_length</i>	The overlap between the shapes must be less than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Overlap Constraints

Example

The overlap distance for Metal1 and Metal2 shapes must be less than or equal to 0.05.

```
orderedSpacings (  
    (maxOverlap "Metal1" "Metal2" 0.05)  
) ;orderedSpacings
```

overlapNotAllowed

```
spacings(  
    ( overlapNotAllowed tx_layer  
      'overlapLayer tx_overlapLayer  
      ['overlapPurpose tx_overlapPurpose]  
      ['maxVoltage f_maxVoltage]  
    )  
); spacings)
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the layer on which overlap is not allowed.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

<i>'overlapLayer tx_overlapLayer</i>	The layer on which overlaps are not allowed. Type: String (layer name) or Integer (layer number)
<i>'overlapPurpose tx_overlapPurpose</i>	The purpose on which overlaps are not allowed.
<i>'maxVoltage f_maxVoltage</i>	The constraint applies only if the voltage on the specified layer shape is greater than this value.

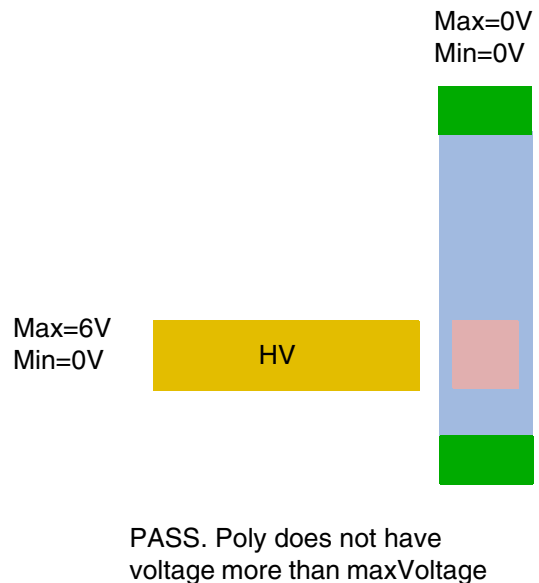
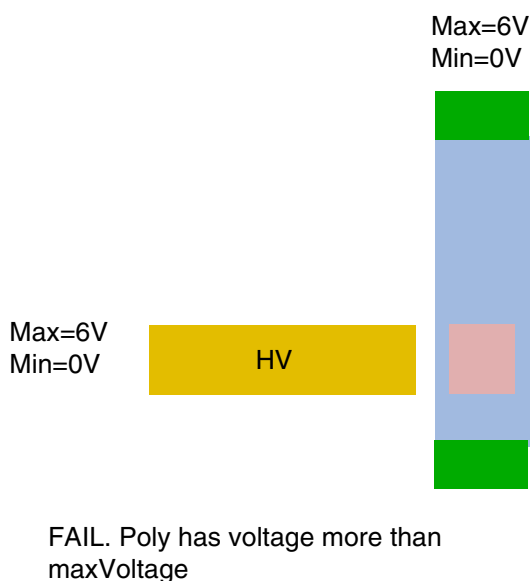
Virtuoso Technology Data Constraints Reference

Overlap Constraints

Example

```
spacings (
  ( overlapNotAllowed "poly"
    'overlapLayer      "gatedNac"
    'overlapPurpose    "gatedNacPurpose"
    'maxVoltage        4
  )
) ;spacings
```

Poly
GatedNAC
Metal
Viag



Packaging Constraints

This topic includes the following constraints:

- pkgMinAperture
- pkgMinApertureDefault
- pkgMinArea
- pkgMinAreaDefault
- pkgMaxBondwireLength
- pkgMinBondwireLength
- pkgMinBondwireSpacing
- pkgMinBondwireWidth
- pkgMinHoleSpacing
- pkgMaxLineWidth
- pkgMinLineWidth
- pkgMinSpacing

pkgMinAperture

```
spacings(  
    ( pkgMinAperture tx_layer  
      f_width  
    )  
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum width of an effective shape or aperture created by voiding a dynamic shape.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_width</i>	The minimum width of the aperture.

Parameters

None.

Example

The minimum width of the apertures on Metal1 is 0.001.

```
spacings(  
    ( pkgMinAperture "Metal1"  
      0.001  
    )  
) ;spacings
```

pkgMinApertureDefault

```
spacings (
    ( pkgMinApertureDefault
      f_width
    )
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum width of an effective shape or aperture created by voiding a dynamic shape. The constraint applies to all layers.

Values

f_width The minimum width of the aperture.

Parameters

None.

Example

The minimum width of the apertures on all layers is 0.001.

```
spacings (
    ( pkgMinApertureDefault
      0.001
    )
) ;spacings
```

pkgMinArea

```
spacings(  
    ( pkgMinArea tx_layer  
      f_area  
    )  
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the square root of the minimum area of an effective shape created by voiding a dynamic shape.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_area</i>	The minimum area in square root.

Parameters

None.

Example

The square root of the minimum area on Metal1 is 0.002.

```
spacings(  
    ( pkgMinArea "Metal1"  
      0.002  
    )  
) ;spacings
```


pkgMinAreaDefault

```
spacings (
    ( pkgMinAreaDefault
      f_area
    )
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the square root of the minimum area of an effective shape created by voiding a dynamic shape. The constraint applies to all layers.

Values

f_area The minimum area in square root.

Parameters

None.

Example

The square root of the minimum default area for all layers is 0.002.

```
spacings (
    ( pkgMinAreaDefault
      0.002
    )
) ;spacings
```

pkgMaxBondwireLength

```
spacings (
    ( pkgMaxBondwireLength
      'profileName t_profileName
      f_length
    )
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the maximum length for bond wires of the specified profile.

Values

f_length The maximum length.

Parameters

'profileName t_profileName
The constraint applies to bond wires of this profile.

Example

The maximum length for the bond wires of the prf2 profile is 0.13.

```
spacings (
    ( pkgMaxBondwireLength
      'profileName "prf2"
      0.13
    )
) ;spacings
```

pkgMinBondwireLength

```
spacings (
    ( pkgMinBondwireLength
      'profileName t_profileName
      f_length
    )
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum length for the bond wires of the specified profile.

Values

<i>f_length</i>	The minimum length.

Parameters

<i>'profileName t_profileName</i>	The constraint applies to the bond wires with this profile.

Example

The minimum length for the bond wires of the prf2 profile is 0.05.

```
spacings (
    ( pkgMinBondwireLength
      'profileName "prf2"
      0.05
    )
) ;spacings
```

pkgMinBondwireSpacing

```
spacings (
    ( pkgMinBondwireSpacing
      'profileName t_profileName
      'bondWire | 'bondFinger | 'pad
      f_spacing
    )
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum spacing for bond wires to objects of the specified type.

Values

f_spacing The minimum spacing.

Parameters

'profileName *t_profileName*

The constraint applies to bond wires with this profile.

'bondWire | 'bondFinger | 'pad

The spacing is applied to objects of this type.

Example

The minimum spacing for the bond wires of the prf1 profile and bondWire object type is 0.11.

```
spacings (
    ( pkgMinBondwireSpacing
      'profileName "prf1"
      'bondWire
      0.11
    )
) ;spacings
```

pkgMinBondwireWidth

```
spacings (
    ( pkgMinBondwireWidth
      'profileName t_profileName
      f_width
    )
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum width for bond wires of the specified profiles.

Values

f_width The minimum width.

Parameters

'profileName t_profileName
The constraint applies to bond wires with this profile.

Example

The minimum width for the bond wires of the prf2 profile is 0.1.

```
spacings (
    ( pkgMinBondwireWidth
      'profileName "prf2"
      0.1
    )
) ;spacings
```

pkgMinHoleSpacing

```
spacings (
    ( pkgMinHoleSpacing tx_layer
      'line | 'pin | 'thruPin | 'smdPin | 'testPin | 'thruVia | 'bbVia |
      'testVia | 'microVia | 'shape | 'bondFinger | 'via | 'hole
      'diffNet | 'sameNet
      f_spacing
    )
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum spacing between drill shapes and shapes of the specified object type on the specified layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing.

Parameters

'line 'pin 'thruPin 'smdPin 'testPin 'thruVia 'bbVia 'testVia 'microVia 'shape 'bondFinger 'via 'hole	The constraint applies only to objects of this type.
'diffNet 'sameNet	The constraint applies only to shapes on different nets or the same net, depending on the argument specified.

Example

The minimum spacing between drill shapes and shapes that are on different nets on the Metal1 layer of the bondFinger object type is 0.1.

```
spacings (
    ( pkgMinHoleSpacing "Metal1"
      'bondFinger
      'diffNet
      0.1
    )
) ;spacings
```

pkgMaxLineWidth

```
spacings(  
    ( pkgMaxLineWidth tx_layer  
      f_width  
    )  
); spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the maximum line width for traces in a package design.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_width</i>	The maximum width.

Parameters

None.

Example

The maximum line width for traces on the Metal1 layer is 0.02.

```
spacings(  
    ( pkgMaxLineWidth "Metal1"  
      0.02  
    )  
); spacings
```

pkgMinLineWidth

```
spacings(  
    ( pkgMinLineWidth tx_layer  
      f_width  
    )  
) ;spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum line width for traces in a package design.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_width</i>	The minimum width.

Parameters

None.

Example

The minimum line width for traces on the Metal1 layer is 0.001.

```
spacings(  
    ( pkgMinLineWidth "Metal1"  
      0.001  
    )  
) ;spacings
```


Virtuoso Technology Data Constraints Reference

Packaging Constraints

pkgMinSpacing

```
spacings(  
    ( pkgMinSpacing tx_layer  
        {'line' | 'pin' | 'thruPin' | 'smdPin' | 'testPin' | 'thruVia' | 'bbVia' |  
          'testVia' | 'microVia' | 'shape' | 'bondFinger' | 'via' | 'hole'}  
        {'lineOther' | 'pinOther' | 'thruPinOther' | 'smdPinOther' | 'testPinOther' |  
          'thruViaOther' | 'bbViaOther' | 'testViaOther' | 'microViaOther' |  
          'shapeOther' | 'bondFingerOther' | 'viaOther'}  
        'diffNet' | 'sameNet'  
        f_spacing  
    )  
); spacings
```

(Virtuoso Layout Suite MXL Only) Specifies the minimum spacing between shapes of the specified object types on the specified layer for RF and advanced package designs.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing.

Virtuoso Technology Data Constraints Reference

Packaging Constraints

Parameters

'line | 'pin | 'thruPin | 'smdPin | 'testPin | 'thruVia | 'bbVia |
'testVia | 'microVia | 'shape | 'bondFinger | 'via | 'hole

The constraint applies only to objects of this type.

'lineOther | 'pinOther | 'thruPinOther | 'smdPinOther |
'testPinOther | 'thruViaOther | 'bbViaOther | 'testViaOther |
'microViaOther | 'shapeOther | 'bondFingerOther | 'viaOther

The constraint applies only with objects of this type.

'diffNet | 'sameNet

The constraint applies only to shapes on different nets or the same net, depending on the argument specified.

Example

The minimum spacing between two Metal1 via shapes on different nets is 0.1.

```
spacings(  
  ( pkgMinSpacing "Metal1"  
    'via 'viaOther  
    'diffNet  
    0.1  
  )  
) ;spacings
```

Placement and Alignment Constraints

This topic includes the following constraints:

- allowedWidthSpacingPatternGroups
- allowedWidthSpacingPatterns
- defaultActiveWidthSpacingPattern
- horizontalOffset
- horizontalPitch
- keepPRBoundarySharedEdges
- keepSharedEdges
- snapGridHorizontal
- snapGridVertical
- verticalOffset
- verticalPitch

allowedWidthSpacingPatternGroups

```
spacings (
    ( allowedWidthSpacingPatternGroups
      'snapPatternDef t_spDefName
      (l_patternNames)
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the width spacing pattern groups allowed in a width spacing snap pattern definition.

Values

l_patternNames A list of allowed width spacing pattern group names.

Parameters

'snapPatternDef t_spDefName

The width spacing snap pattern definition to which this constraint applies.

Example

The width spacing pattern groups allowed in width spacing snap pattern definition `m1_grid` are `1X_group` and `basePatterns`.

```
spacings (
    ( allowedWidthSpacingPatternsGroups
      'snapPatternDef "m1_grid"
      ("1X_group" "basePatterns")
    )
) ;spacings
```

allowedWidthSpacingPatterns

```
spacings (
    ( allowedWidthSpacingPatterns
      'snapPatternDef t_snapPatternDefName
      (l_patternNames)
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the width spacing patterns allowed in a width spacing snap pattern definition.

Values

l_patternNames A list of allowed width spacing pattern names.

Parameters

'snapPatternDef *t_snapPatternDefName*

The width spacing snap pattern definition to which the constraint applies.

Example

The width spacing patterns allowed in width spacing snap pattern definition `m1_grid` are 1X and 2X.

```
spacings (
    ( allowedWidthSpacingPatterns
      'snapPatternDef "m1_grid"
      ("1X" "2X")
    )
) ;spacings
```

defaultActiveWidthSpacingPattern

```
spacings (
    ( defaultActiveWidthSpacingPattern
      'snapPatternDef t_spdefname
      t_patternName
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the default active pattern for a width spacing snap pattern definition.

Values

t_patternName The name of the default active width spacing pattern.

Parameters

'snapPatternDef *t_spdefname*

The width spacing snap pattern definition to which the constraint applies.

Example

The default active pattern for width spacing snap pattern definition `m1_grid` is `1X`.

```
spacings (
    (defaultActiveWidthSpacingPattern
      'snapPatternDef "m1_grid"
      "1X"
    )
) ;spacings
```

horizontalOffset

```
placementGrids(  
    ( horizontalOffset f_offset )  
);placementGrids
```

Sets the minimum offset from the origin for the vertical placement grid.

Values

<i>f_offset</i>	The horizontal offset from the origin for the vertical placement grid. The default value is 0.0.

Parameters

None

Example

The vertical placement grid must be offset by 1.3 from the origin.

```
placementGrids(  
    ( horizontalOffset 1.3 )  
);placementGrids
```

horizontalPitch

```
placementGrids(  
    ( horizontalPitch f_pitch )  
);placementGrids
```

Sets the minimum spacing between vertical placement grid lines.

Values

<i>f_pitch</i>	The minimum spacing between vertical placement grid lines.

Parameters

None

Example

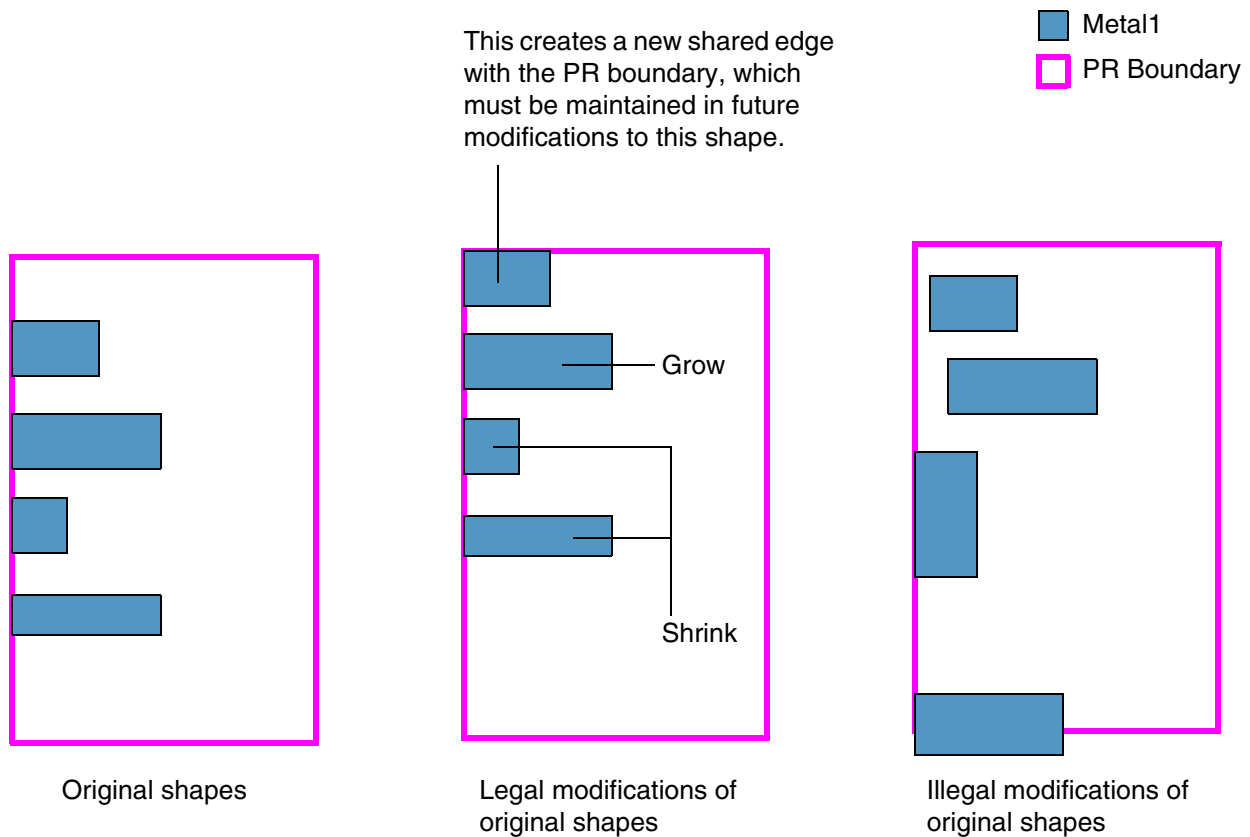
The minimum spacing required between vertical placement grid lines is 1.0.

```
placementGrids(  
    ( horizontalPitch 1.0 )  
);placementGrids
```


keepPRBoundarySharedEdges

```
spacings (
  ( keepPRBoundarySharedEdges tx_layer
    {t | nil}
  )
) ; spacings
```

Specifies that shapes with an edge on the PR boundary must maintain that shared edge when they are modified. All shapes on the specified layer must be entirely inside the PR boundary and share at least one edge with the PR boundary.



Virtuoso Technology Data Constraints Reference

Placement and Alignment Constraints

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>t nil</code>	If <code>t</code> , the shared edges with the PR boundary must be maintained; if <code>nil</code> , the shared edges with the PR boundary need not be maintained.

Parameters

None

Example

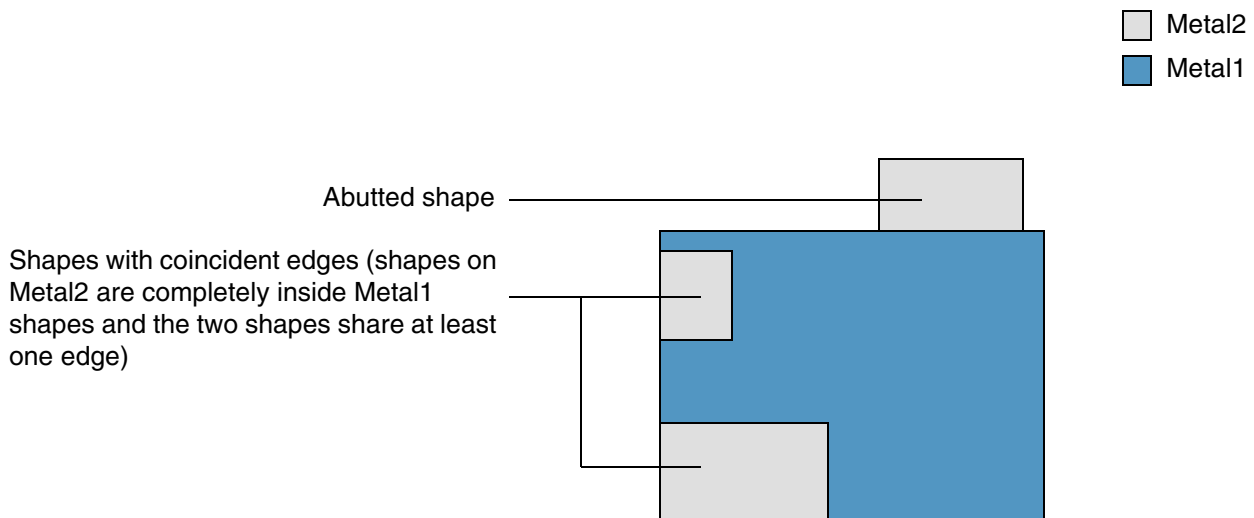
Edges of Metal1 shapes coincident with the PR boundary must remain coincident when the shapes are modified.

```
spacings (
  ( keepPRBoundarySharedEdges "Metal1"
    t
  )
) ;spacings
```

keepSharedEdges

```
spacings (
  ( keepSharedEdges tx_layer1 tx_layer2
    {t | nil}
    ['buttOnly | 'coincidentOnly]
  )
) ;spacings
```

Specifies that abutted or coincident edges of shapes on *layer1* and *layer2* must be maintained when the shapes are modified. The length of the shared edge can change as long as the shapes maintain at least one shared point on the shared edge.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t nil</i>	If <i>t</i> , the shared edges must be maintained; if <i>nil</i> , the shared edges need not be maintained.

Virtuoso Technology Data Constraints Reference

Placement and Alignment Constraints

Parameters

'buttOnly | 'coincidentOnly

The alignment of shapes. By default, the constraint applies to both abutted and coincident edges.

- 'buttOnly: The constraint applies only to abutted edges.
- 'coincidentOnly: The constraint applies only to coincident edges.

Example

Shared edges of Metal1 and Metal2 abutted shapes must remain coincident when the shapes are modified.

```
spacings(  
    ( keepSharedEdges "Metal1" "Poly1"  
      t  
      'buttOnly  
    )  
); spacings
```

snapGridHorizontal

```
spacings (
    ( snapGridHorizontal (l_snapPatternDefNames) )
) ;spacings
```

Enables the global track grid by specifying a list of global snapPatternDefs or widthSpacingSnapPatternDefs. The direction for the specified pattern definitions must be horizontal.

Values

l_snapPatternDefNames

A list of global snapPatternDefs or widthSpacingSnapPatternDefs.

Parameters

None

Example

The global track grid is defined using the poly_grid and m3_grid snapPatternDefs.

```
spacings (
    (snapGridHorizontal ("poly_grid" "m3_grid") )
) ;spacings
```

snapGridPeriod

```
spacings (
    ( snapGridPeriod 'snapPatternDef  t_spDefName  n_period )
) ;spacings
```

Specifies the period for a snapPatternDef.

Values

<i>t_spDefName</i>	The name of the snapPatternDef to which the constraint applies.
<i>n_period</i>	The period to be used.

Parameters

None

Example

Specifies the period of snapPatternDef with name GFG as 4.

```
spacings (
    (snapGridPeriod 'snapPatternDef "GFG" 4)
) ;spacings
```

snapGridVertical

```
spacings (
    ( snapGridVertical (l_snapPatternDefNames) )
) ;spacings
```

Enables the global track grid by specifying a list of global snapPatternDefs or widthSpacingSnapPatternDefs. The direction for the specified pattern definitions must be vertical.

Values

l_snapPatternDefNames

A list of global snapPatternDefs or widthSpacingSnapPatternDefs.

Parameters

None

Example

The global track grid is defined using the finGrid and m2_grid snapPatternDefs.

```
spacings (
    ( snapGridVertical ("finGrid" "m2_grid") )
) ;spacings
```

verticalOffset

```
placementGrids(  
    ( verticalOffset g_offset )  
);placementGrids
```

Sets the minimum offset from the origin for the horizontal placement grid.

Parameters

<i>g_offset</i>	The vertical offset from the origin for the horizontal placement grid. The default value is 0.0.

Example

The horizontal placement grid must be offset by 1.4 from the origin.

```
placementGrids(  
    ( verticalOffset 1.4 )  
);placementGrids
```


verticalPitch

```
placementGrids(  
    ( verticalPitch f_pitch )  
);placementGrids
```

Sets the minimum spacing between horizontal placement grid lines.

Values

<i>f_pitch</i>	The minimum spacing between horizontal placement grid lines.

Parameters

None

Example

The minimum spacing required between horizontal placement grid lines is 1.0.

```
placementGrids(  
    ( verticalPitch 1.0 )  
);placementGrids
```

Virtuoso Technology Data Constraints Reference

Placement and Alignment Constraints

Routing Constraints

This topic includes the following constraints:

- [clusterDistance](#)
- [excludeLPPs](#)
- [horizontalOffset](#)
- [horizontalPitch \(One layer\)](#)
- [ladderFrequency](#)
- [ladderOffset](#)
- [ladderStyle](#)
- [layerMaskShiftAllowed](#)
- [leftDiagOffset](#)
- [leftDiagPitch](#)
- [maxRoutingDistance](#)
- [numStrands](#)
- [orthogonalSnappingLayer](#)
- [orthogonalWSPGrid](#)
- [rectangularGapMinSpacing](#)
- [rightDiagOffset](#)
- [rightDiagPitch](#)
- [routingDirections](#)
- [shieldGroupStyle](#)
- [shieldMinLength](#)

Virtuoso Technology Data Constraints Reference

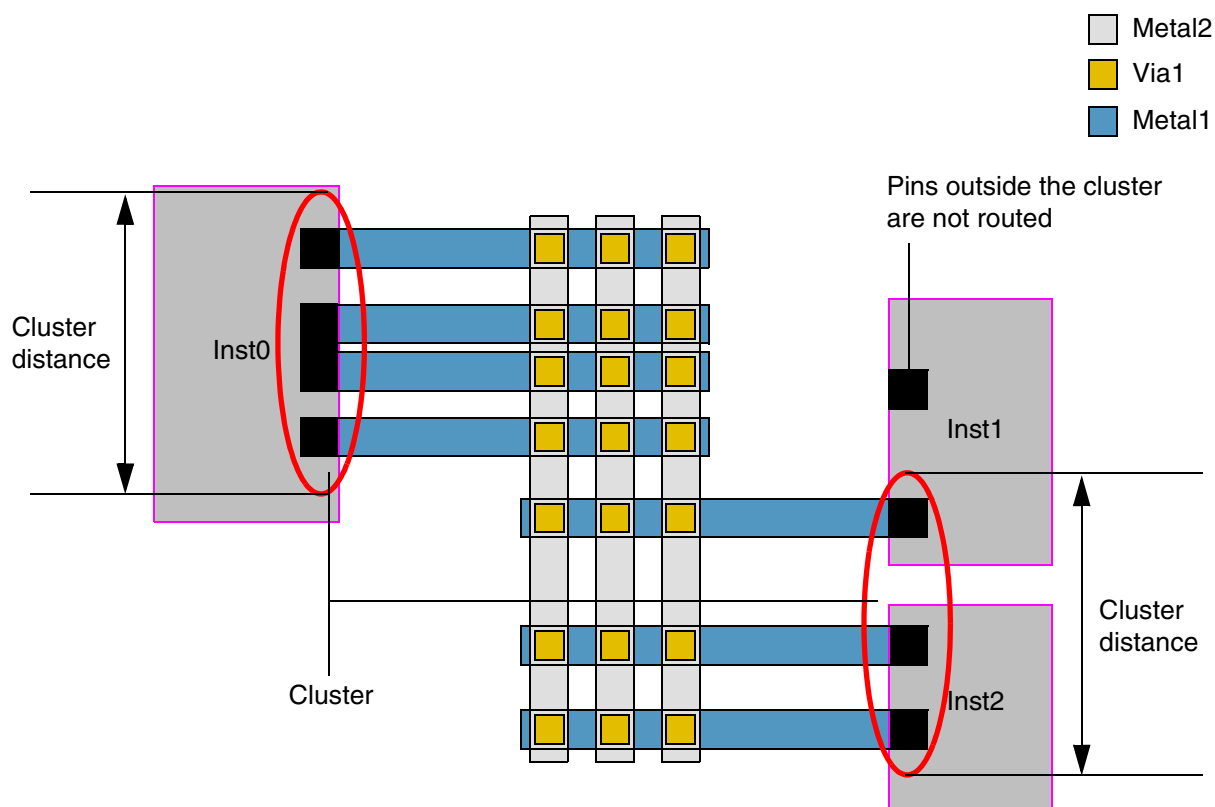
Routing Constraints

- shieldTerminal
- strandNumber
- strandSpacing
- strandWidth
- styleAroundVia
- tieShield
- trackPattern
- taperHalo
- validLayers
- validPurposes
- validVias
- verticalOffset
- verticalPitch (One layer)
- voltageLayerMarkerMapping
- voltageLayerPairMarkerMapping

clusterDistance

```
spacings (
    ( clusterDistance f_clusterDistance )
) ; spacings
```

Specifies that pins at a distance less than or equal to *clusterDistance* from each other on a multi-pin net are clustered together for strand routing.



Values

f_clusterDistance The cluster distance.

Parameters

None

excludeLPPs

```
interconnect(  
    ( excludeLPPs (t_layer t_purpose) ... )  
) ;interconnect
```

Excludes the specified layer-purpose pairs from consideration. A layer-purpose pair is excluded even if that layer-purpose pair is listed in validLayers.

Values

t_layer t_purpose

Layer-purpose pair to be excluded.

Type: A set of valid layer-purpose pairs, each pair enclosed in parentheses and separated by a space

Parameters

None

Example

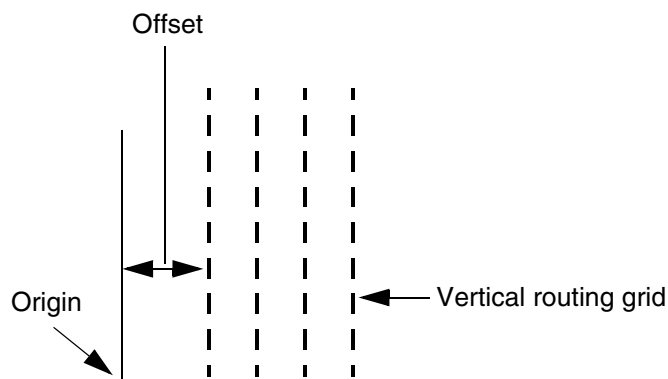
For layers M0, M1, M2, and M3, all layer-purpose pairs with purpose "drawing", "vdd", "vss", "test1", and "test2" are included in connectivity extraction, except layer-purpose pairs "M0, test1" and "M1, test2".

```
("virtuosoDefaultExtractorSetup" nil  
  interconnect(  
    ( validLayers ("M0" "M1" "M2" "M3") )  
    ( validPurposes 'include ("drawing" "vdd" "vss" "test1" "test2") )  
    ( excludeLPPs ("M0" "test1") ("M1" "test2") )  
  ) ;interconnect  
) ;virtuosoDefaultExtractorSetup
```

horizontalOffset

```
routingGrids(
    ( horizontalOffset [tx_layer] f_offset )
    ...
) ;routingGrids
```

Sets the offset from the origin for the start of the vertical routing grid. A horizontal offset specified with a layer applies only to that layer. When specified without a layer, it sets the default for all layers not explicitly assigned a horizontal offset.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_offset</i>	The offset from the origin for the vertical routing grid. The default value is 0.

Parameters

None

Example

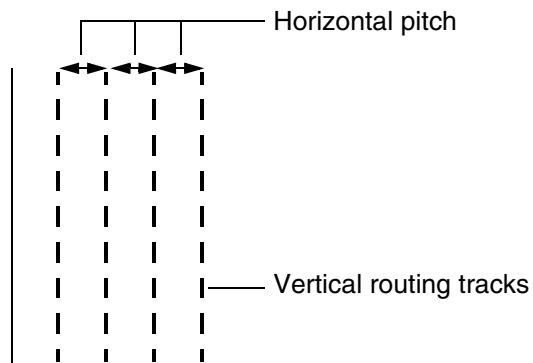
The horizontal offset must be 1.2 for Metal3 and 1.5 for the rest of the layers.

```
routingGrids(
    ( horizontalOffset "Metal3" 1.2 )
    ( horizontalOffset 1.5 )
) ;routingGrids
```

horizontalPitch (One layer)

```
routingGrids (
    ( horizontalPitch [tx_layer]
      ['firstLastPitch f_firstLastPitch]
      f_pitch
    )
) ;routingGrids
```

Specifies the horizontal pitch (distance) between the routing tracks on the specified layer. If *layer* is not specified, the constraint sets the default pitch for all layers for which a horizontal pitch is not explicitly assigned.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_pitch</i>	The distance between the vertical routing tracks on the specified layer. If 'firstLastPitch' is specified, this distance applies only to the tracks between the first and last routing tracks.

Parameters

'firstLastPitch f_firstLastPitch'	The distance of the first and the last routing track from the cell row boundary.

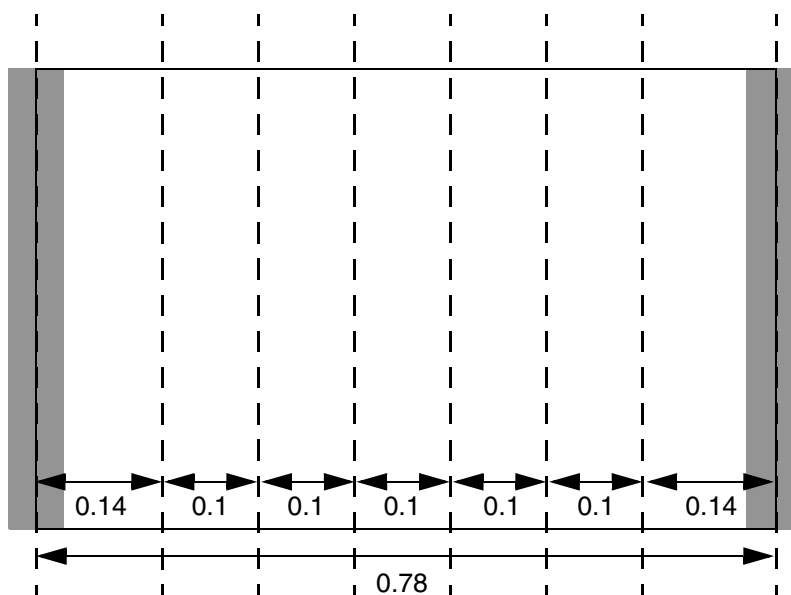
Virtuoso Technology Data Constraints Reference

Routing Constraints

Example

The horizontal pitch is 0.14 for the first and last routing tracks and 0.1 for intermediate tracks. This applies to all layers for which horizontal pitch is not explicitly assigned.

```
routingGrids(  
  ( horizontalPitch  
    'firstLastPitch 0.14  
    0.1  
  )  
) ;routingGrids
```



The dotted lines represent vertical routing tracks. The first and last tracks are 0.14 away from the cell row boundary and the intermediate tracks are 0.1 apart. The cell row width is 0.78 ($2 \times 0.14 + 5 \times 0.1$).

ladderFrequency

```
spacings (  
    ( ladderFrequency f_frequency )  
); spacings
```

Specifies the percentage of laddering. A 100 percent is equivalent to the maximum amount of laddered routing (`minSpacing`) and a zero percent is equivalent to laddering at only the turns.

Values

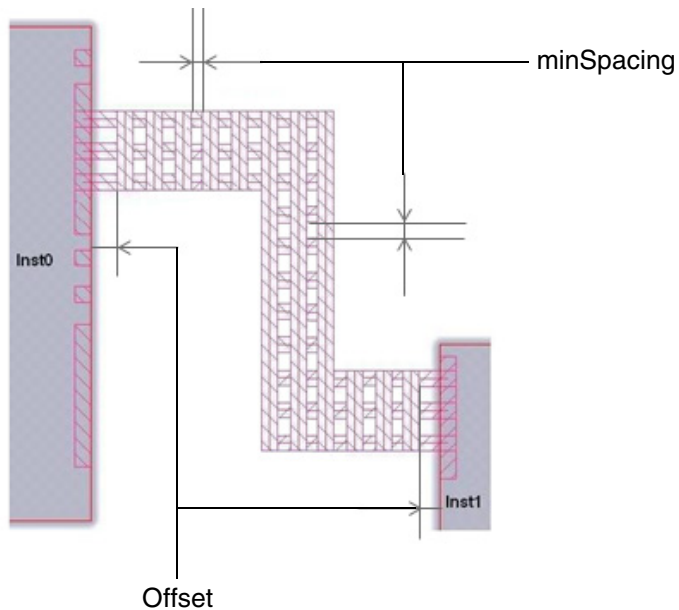
f_frequency The laddering percentage.

Parameters

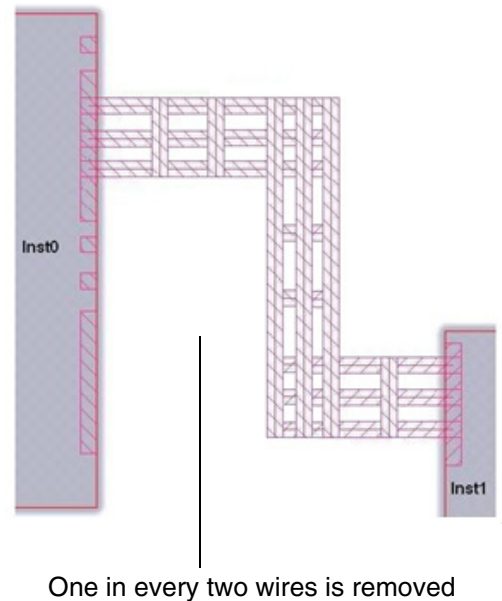
None

Example

Frequency = 100%



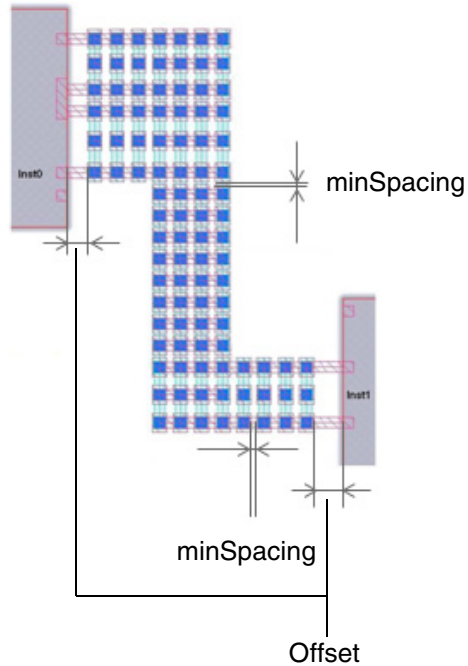
Frequency = 50%



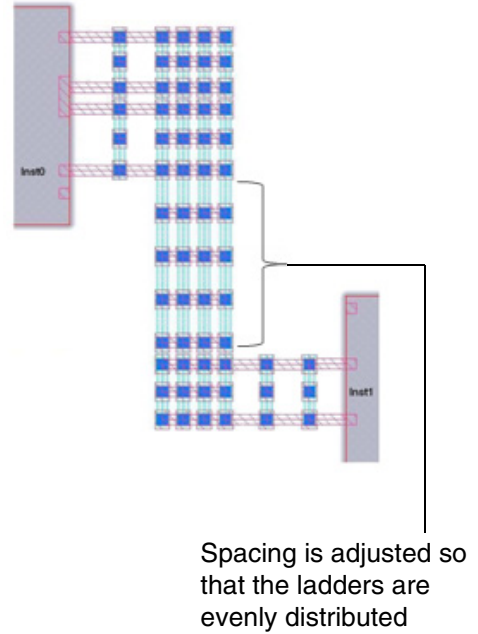
Virtuoso Technology Data Constraints Reference

Routing Constraints

Frequency = 100%



Frequency = 50%



ladderOffset

```
spacings(  
    ( ladderOffset tx_layer f_offset )  
); spacings
```

Specifies the distance between the pin and the first portion of the ladder.

Values

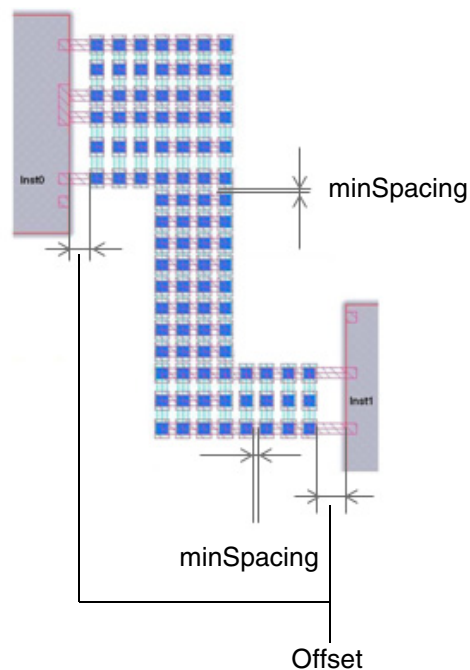
<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_offset</i>	The offset.

Parameters

None

Example

Frequency = 100%



ladderStyle

```
spacings (
    ( ladderStyle
        ['sameLayerLadder | 'layerAboveLadder | 'layerBelowLadder]
        {"noLadderStyle" | "atTurnOnly" | "allSegments"}
    )
) ;spacings
```

Specifies the laddering style for strand routing.

Values

"noLadderStyle" | "atTurnOnly" | "allSegments"

The ladder style.

- **noLadderStyle:** No additional perpendicular wires are added to the wire strands.
- **atTurnOnly:** Additional perpendicular wires are added only at a turn, if there exists one.
- **allSegments:** Additional perpendicular wires are added to entire wires.

Parameters

'sameLayerLadder | 'layerAboveLadder | 'layerBelowLadder

The layer to be used for laddering.

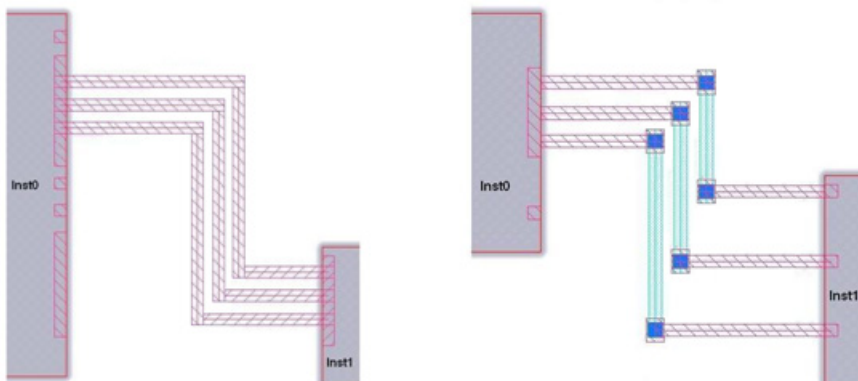
- **'sameLayerLadder:** The wire strands and the associated perpendicular wires are on the same layer.
- **'layerAboveLadder:** The associated perpendicular wires are on the layer above the wire strands.
- **'layerBelowLadder:** The associated perpendicular wires are on the layer below the wire strands.

Virtuoso Technology Data Constraints Reference

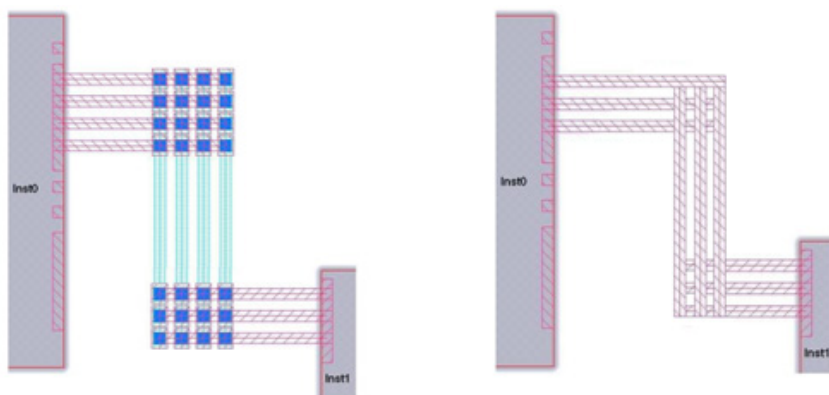
Routing Constraints

Examples

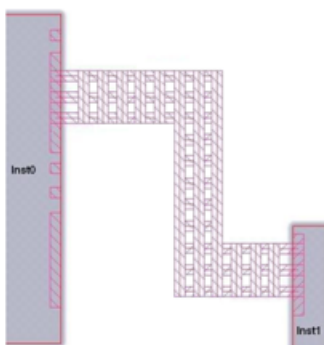
noLadderStyle



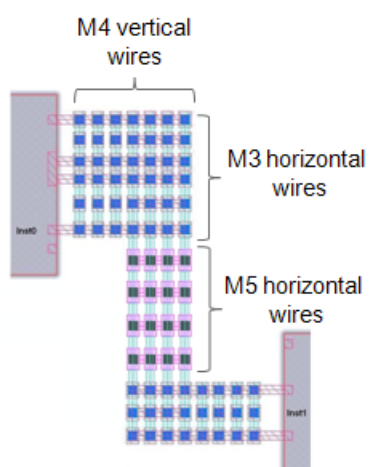
atTurnOnly



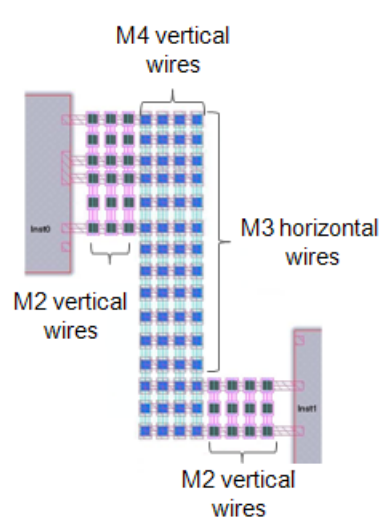
allSegments



sameLayerLadder



layerAboveLadder



layerBelowLadder

layerMaskShiftAllowed

```
spacings (
    ( layerMaskShiftAllowed (l_layerNames) )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the layers on which masks can be shifted.

Values

l_layerNames The layers on which the constraint is applied.
Type: A list of layer names

Example

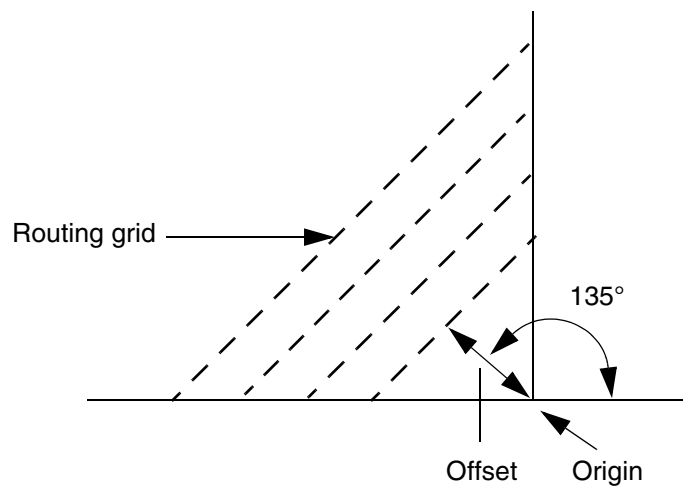
Mask shifting is allowed only on layers Metal1 through Metal4 and Via1.

```
spacings (
    ( layerMaskShiftAllowed ("Metal1" "Metal2" "Metal3" "Metal4" "Via1") )
) ;spacings
```

leftDiagOffset

```
routingGrids(  
    ( leftDiagOffset [tx_layer] f_offset )  
    ...  
) ;routingGrids
```

Sets the offset from the origin for the start of the left diagonal routing grid. A left diagonal offset specified with a layer applies to that layer. When specified without a layer, it sets the default for all layers not explicitly assigned a left diagonal offset.



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_offset

The offset from the origin. The default value is 0.

Parameters

None

Virtuoso Technology Data Constraints Reference

Routing Constraints

Example

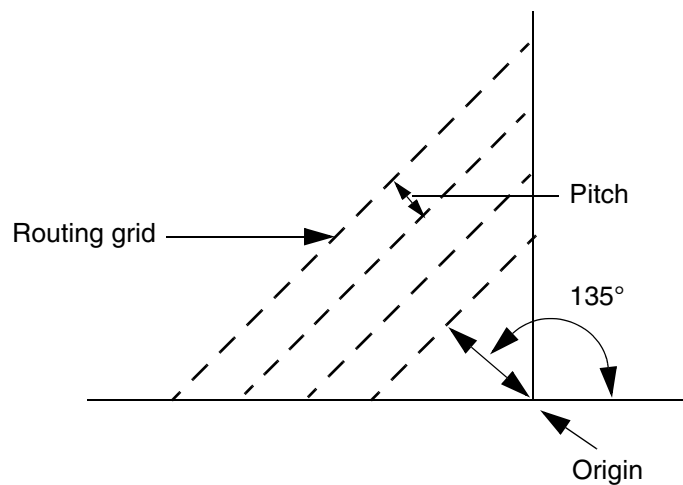
The left diagonal offset must be 1.2 for Metal3 and 1.4 for the rest of the layers.

```
routingGrids(  
    ( leftDiagOffset "Metal3" 1.2 )  
    ( leftDiagOffset 1.4 )  
) ;routingGrids
```

leftDiagPitch

```
routingGrids(  
    ( leftDiagPitch [ tx_layer ] f_pitch )  
    ...  
) ;routingGrids
```

Specifies the left diagonal pitch (distance) between the routing tracks on the specified layer. If layer is not specified, the constraint sets the default pitch for all layers for which a left diagonal pitch is not explicitly assigned.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_pitch</i>	The spacing between the routing grid lines.

Parameters

None

Virtuoso Technology Data Constraints Reference

Routing Constraints

Example

The left diagonal pitch must be 1.0 for Metal3 and 0.8 for the rest of the layers.

```
routingGrids (
  ( leftDiagPitch "Metal3" 1.0 )
  ( leftDiagPitch 0.8 )
) ;routingGrids
```

maxRoutingDistance

```
interconnect(  
    ( maxRoutingDistance tx_layer f_distance )  
    ...  
);interconnect
```

Sets the maximum routing distance, applied on a per-net basis, allowed for the specified layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_distance</i>	The maximum distance.

Parameters

None

Example

The maximum routing distance on Metal3 must be 5.0.

```
interconnect(  
    ( maxRoutingDistance "Metal3" 5.0 )  
);interconnect
```

numStrands

```
spacings(  
    ( numStrands x_strands )  
); spacings
```

Specifies the number of strands used to route between pins.

Values

x_strands Specifies the number of strands used to route between pins.

Parameters

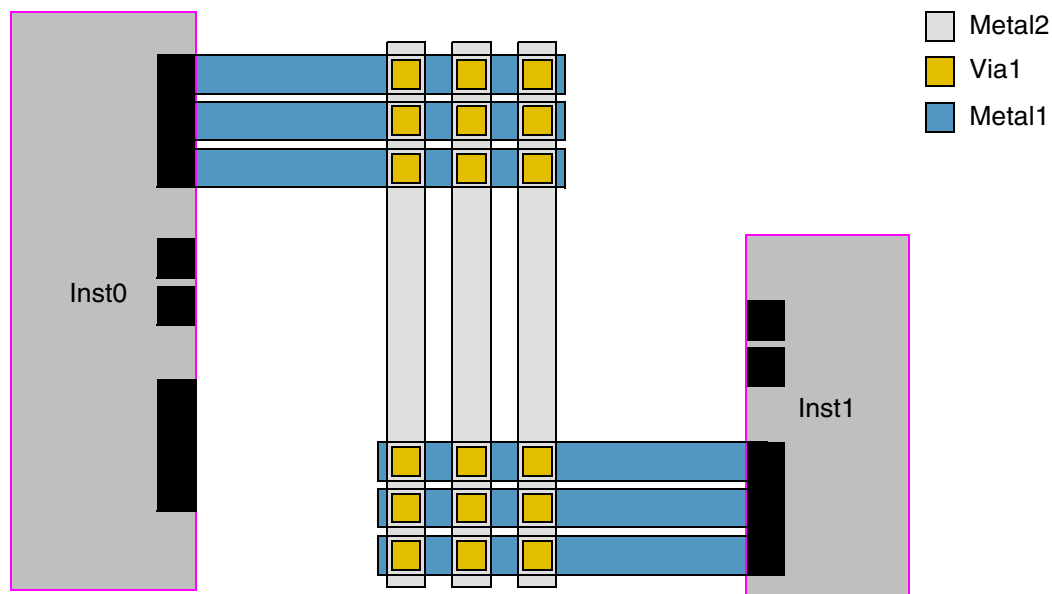
None

Examples

- [Example 1: Fat-to-fat pins](#)
- [Example 2: Fat-to-many pins](#)
- [Example 3: Many-to-many pins](#)

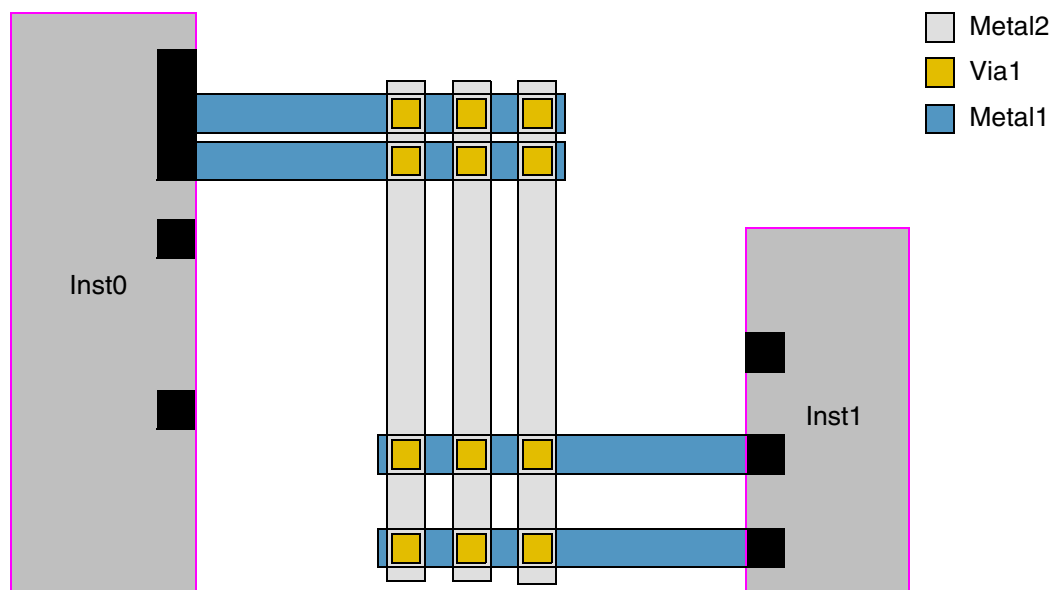
Example 1: Fat-to-fat pins

The pin is split to accommodate the number of ($n=3$) strands.



Example 2: Fat-to-many pins

The fat pin is split to accommodate the number ($n=2$) of strands, and then the n strands are distributed among the pins of the "many" cluster.

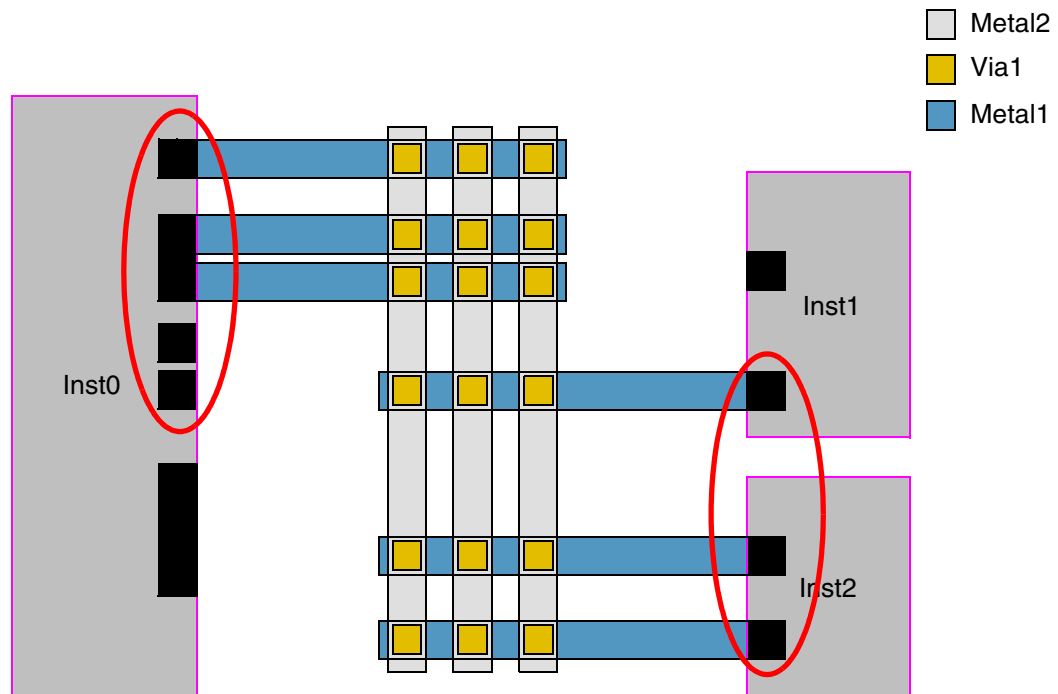


Virtuoso Technology Data Constraints Reference

Routing Constraints

Example 3: Many-to-many pins

The number of strands ($n=3$) is distributed among the pins in the cluster.



orthogonalSnappingLayer

```
spacingTables (
  ( orthogonalSnappingLayer tx_layer1 tx_layer2
    (( "width" nil nil )
      ['upperLineEndStoppingPoints g_upperTable]
      ['mask1 | 'mask2 | 'mask3]
      ['otherMask1 | 'otherMask2 | 'otherMask3]
      l_defaultRanges
    )
    g_rangeTable
  )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Only) 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.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. The constraint applies to the line-end edges on this layer. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. The line-end edges on <i>layer1</i> need to align to the grid on this layer. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil	This identifies the index for <i>rangeTable</i> .

Virtuoso Technology Data Constraints Reference

Routing Constraints

g_rangeTable

The format of the *rangeTable* row is as follows:

(*f_index* *l_valueRange*)

where,

- *f_index* is the width of a track on *layer2*. The corresponding stopping-point distance values apply if the width of a track is greater than or equal to *index*.
- *l_valueRange* is a list of stopping-point distances relative to the centerline of the tracks on *layer2*. A value of zero means that the end-of-line must stop at the centerline of a track; a negative value means that the end-of-line must stop at the specified distance before the centerline; and a positive value means that the end-of-line must stop at the specified distance after the centerline.

Type: A 1-D table specifying a list of floating-point distance ranges, indexed on the width of the tracks on *layer2*

Parameters

'upperLineEndStoppingPoints *g_upperTable*

Each entry in the table specifies 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: A 1-D table specifying floating-point distance ranges, indexed on the width of the routing tracks on *layer2*

'mask1 | 'mask2 | 'mask3

The constraint applies to this mask on *layer1*.

'otherMask1 | 'otherMask2 | 'otherMask3

The constraint applies to this mask on *layer2*.

l_defaultRanges

The stopping-point distance values to be used when no table entry applies.

Type: A list of stopping-point distances ranges

Virtuoso Technology Data Constraints Reference

Routing Constraints

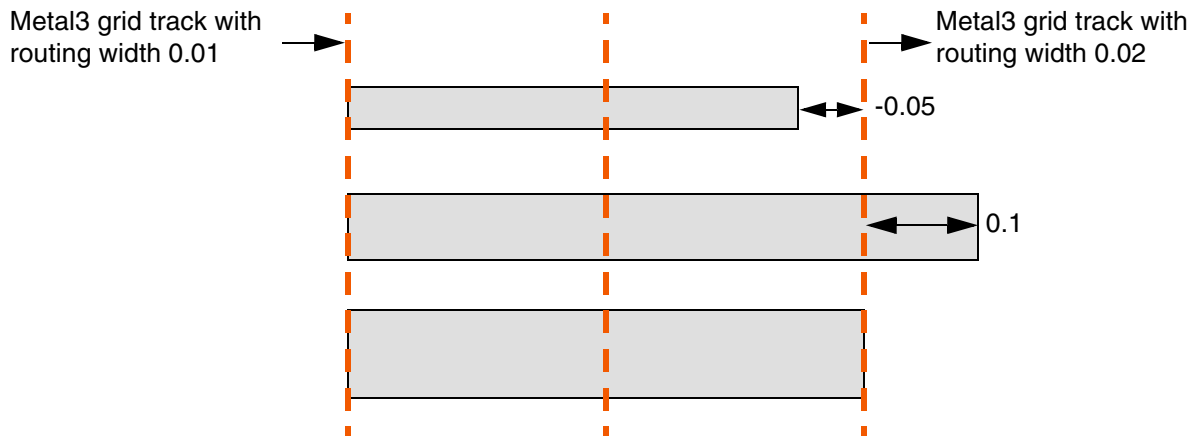
Example

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.

```
spacingTables (  
  ( orthogonalSnappingLayer "Metal2" "Metal3"  
    (( "width" nil nil ))  
    (  
      0.00 (0)  
      0.02 (-0.05 0 0.1)  
    )  
  )  
) ;spacingTables
```

■ Metal3
■ Metal2



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.

orthogonalWSPGrid

```
spacings (
    ( orthogonalWSPGrid tx_layer
      ['upperLineEndWSSPDefName t_upperLineEndWsspDefName]
      ['mask1 | 'mask2 | 'mask3]
      t_lineEndWSSPDefName
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies one or two widthSpacingSnapPatternDef values that define the line-end grid for a routing layer. The line-ends on the given layer must coincide with the grid defined by the defaultActive WSP of the widthSpacingSnapPatternDef.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t_lineEndWSSPDefName</i>	The widthSpacingSnapPatternDef name.

Parameters

<i>'upperLineEndWSSPDefName t_upperLineEndWsspDefName</i>	The specified value is used for right line-ends of horizontal wires and top line-ends of vertical wires. For left line-ends of horizontal wires and bottom line-ends of vertical wires, the constraint value is used. If not specified, the constraint value is used for all line-ends.

Virtuoso Technology Data Constraints Reference

Routing Constraints

'mask1 | 'mask2 | 'mask3

The constraint applies only to the shapes on this mask. The color of the tracks in the WSP grid referenced by the constraint is ignored.

Type: Boolean

Examples

- Example 1: orthogonalWSPGrid with upperLineEndWSSPDefName
- Example 2: orthogonalWSPGrid with upperLineEndWSSPDefName and mask1

Example 1: orthogonalWSPGrid with upperLineEndWSSPDefName

The upper line-ends of Metal2 shapes must snap to M2WSP_upper and the lower line-ends of Metal2 shapes must snap to M2WSP.

```
spacings(  
  ( orthogonalWSPGrid "Metal2"  
    'upperLineEndWSSPDefName "M2WSP_upper"  
    "M2WSP"  
  )  
); spacings
```

Example 2: orthogonalWSPGrid with upperLineEndWSSPDefName and mask1

The upper line-ends of Metal2 mask1 shapes must snap to M2WSP_upper and the lower line-ends of Metal2 mask1 shapes must snap to M2WSP_lower.

```
spacings(  
  ( orthogonalWSPGrid "Metal2"  
    'upperLineEndWSSPDefName "M2WSP_upper"  
    'mask1  
    "M2WSP_lower"  
  )  
); spacings
```

rectangularGapMinSpacing

```
spacingTables (
  ( rectangularGapMinSpacing tx_layer
    (( "track" nil nil )
      'exactWidth f_exactWidth 'maxLength f_maxLength
      ['endToEndSpacing f_endToEndSpacing]
    )
    (g_table)
  )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Only) Specifies the dimensions and spacing requirements for rectangular gaps on a routing layer. No wires on the routing layer can be inside these gaps.

Some processes require the entire design to be filled by wires, except for some rectangular gaps with an exact width and with length less than or equal to the specified value. These rectangular gaps must satisfy spacing requirements based on whether the gaps are on the same track, on adjacent tracks, or two tracks apart, where a track is along the preferred direction of the routing layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"track" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The spacing between gaps on the same track (index 0), on adjacent tracks (index 1), and two tracks apart (index 2). Type: A 1-D table specifying floating-point spacing values

Virtuoso Technology Data Constraints Reference

Routing Constraints

Parameters

`'exactWidth f_exactWidth`

The exact width for a gap.

`'maxLength f_maxLength`

The maximum length for a gap.

`'endToEndSpacing f_endToEndSpacing`

The end-to-end spacing between gaps with parallel run length greater than zero must be greater than or equal to this value.

Examples

- Example 1: rectangularGapMinSpacing with exactWidth and maxLength
- Example 2: rectangularGapMinSpacing with exactWidth, maxLength, and endToEndSpacing

Example 1: rectangularGapMinSpacing with exactWidth and maxLength

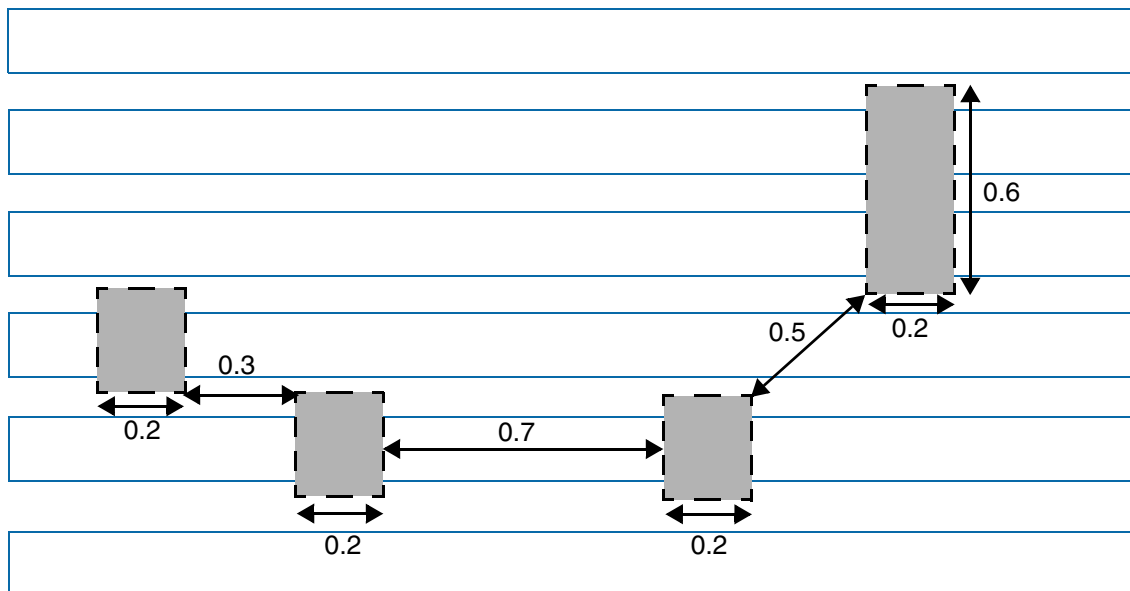
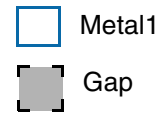
Gaps must have an exact width of 0.2 and must be less than or equal to 0.6 in length. There can be no wiring in these gaps. Spacing between gaps must be:

- Greater than or equal to 0.4 when on the same track
- Greater than or equal to 0.3 when on adjacent tracks
- Greater than or equal to 0.5 when the gaps are two tracks apart

Virtuoso Technology Data Constraints Reference

Routing Constraints

```
spacingTables (
  ( rectangularGapMinSpacing "Metal1"
    (( "track" nil nil )
      'exactWidth 0.2 'maxLength 0.6
    )
  )
  (
    0 0.7
    1 0.3
    2 0.5
  )
) ;spacingTables
```



PASS. The width of the gaps is 0.2 and the length is less than or equal to 0.6. The spacing between the two middle gaps on the same track is 0.7; spacing between the leftmost gaps on adjacent tracks is 0.3; and spacing for the two rightmost gaps that are two tracks apart is 0.5.

Example 2: rectangularGapMinSpacing with exactWidth, maxLength, and endToEndSpacing

Gaps must have an exact width of 0.2 and must be less than or equal to 0.6 in length. Spacing between gaps must be:

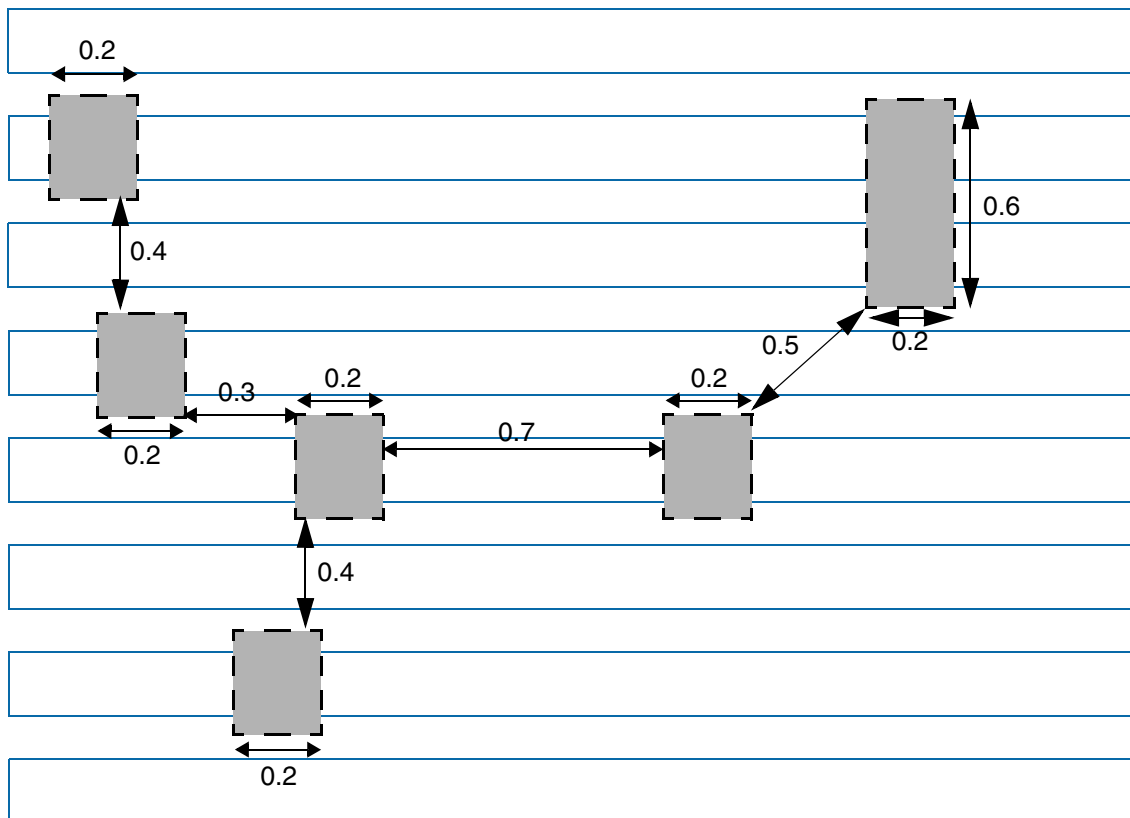
- Greater than or equal to 0.4 when on the same track
- Greater than or equal to 0.3 when on adjacent tracks
- Greater than or equal to 0.5 when the gaps are two tracks apart

Virtuoso Technology Data Constraints Reference

Routing Constraints

The end-to-end spacing between gaps must be greater than or equal to 0.4 if the parallel run length between gaps is greater than zero.

```
spacingTables (
  ( rectangularGapMinSpacing "Metal1"
    (( "track" nil nil )
      'exactWidth 0.2 'maxLength 0.6
      'endToEndSpacing 0.4
    )
  )
  (
    0 0.7
    1 0.3
    2 0.5
  )
)
) ;spacingTables
```

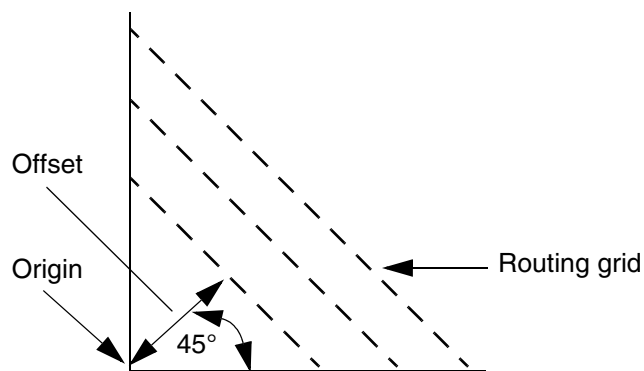


PASS. The width of the gaps is 0.2 and the length is less than or equal to 0.6. The spacing between the two middle gaps on the same track is 0.7; spacing between the leftmost gaps on adjacent tracks is 0.3; spacing for the two rightmost gaps that are two tracks apart is 0.5; and the end-to-end spacing between gaps with parallel run length greater than zero is 0.4.

rightDiagOffset

```
routingGrids(  
    ( rightDiagOffset [tx_layer] f_offset )  
    ...  
) ;routingGrids
```

Sets the offset from the origin for the start of the right diagonal routing grid. A right diagonal offset specified with a layer applies to that layer. When specified without a layer, it sets the default for all layers not explicitly assigned a right diagonal offset.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_offset</i>	The offset from the origin. The default value is 0.

Parameters

None

Example

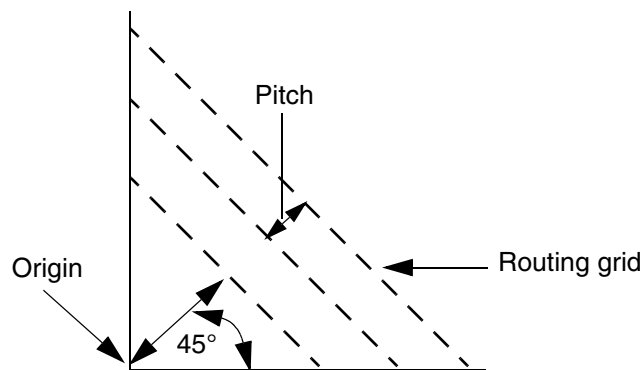
The right diagonal offset must be 1.2 for Metal3 and 1.4 for the rest of the layers.

```
routingGrids(  
    ( rightDiagOffset "Metal3" 1.2 )  
    ( rightDiagOffset 1.4 )  
) ;routingGrids
```

rightDiagPitch

```
routingGrids(  
    ( rightDiagPitch [tx_layer] f_pitch )  
    ...  
) ;routingGrids
```

Specifies the right diagonal pitch (distance) between the routing tracks on the specified layer. If *layer* is not specified, the constraint sets the default pitch for all layers for which a right diagonal pitch is not explicitly assigned.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_pitch</i>	The spacing between the routing grid lines.

Parameters

None

Example

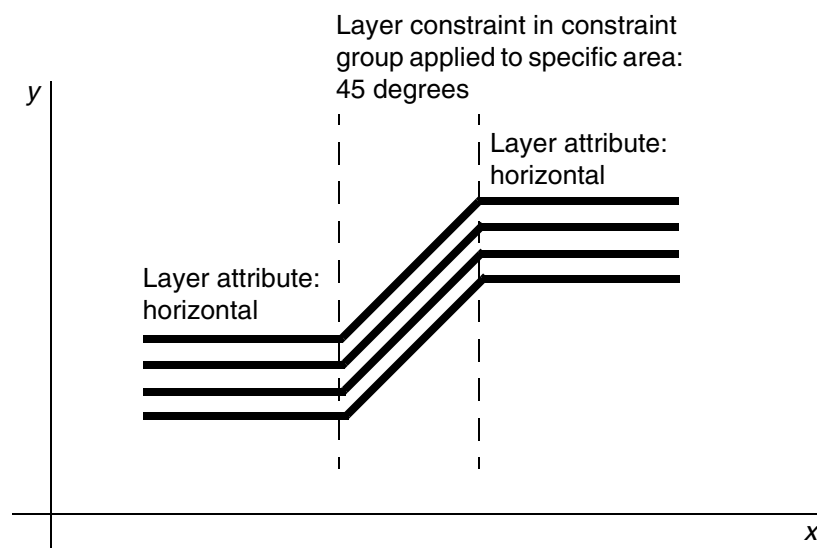
The right diagonal pitch must be 1.0 for Metal3 and 1.2 for the rest of the layers.

```
routingGrids(  
    ( rightDiagPitch "Metal3" 1.0 )  
    ( rightDiagPitch 1.2 )  
) ;routingGrids
```

routingDirections

```
routingDirections(
    ( tx_layer t_direction )
    ...
) ;routingDirections
```

Allows you to override the routingDirections (routing layer direction) layer attribute for a defined section (bounded area) of a design. Applying a constraint group specifying routingDirections to a bounded area redefines the routing directions within that area.



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

t_direction

The routing direction for the layer inside any area to which the constraint group is applied. Valid values include `vertical`, `horizontal`, `leftDiag`, `rightDiag`, and `none`.

- `leftDiag` = 135 degrees
- `rightDiag` = 45 degrees

Parameters

None

Example

Layer attributes are defined as follows:

```
layerRules(  
  routingDirections(  
    ("Metal1" "horizontal")  
    ("Metal2" "vertical")  
    ("Metal3" "rightDiag")  
    ("Diff"   "none")  
    ("Vial"   "notApplicable")  
  );routingDirections  
);layerRules
```

The `routingDirections` constraint group is defined as follows:

```
constraintGroups(  
  ("myRoutingLayerOverrides"  
    routingDirections(  
      ("Metal1" "rightDiag")  
      ("Metal2" "horizontal")  
      ("Metal3" "none")  
    );routingDirections  
  );myRoutingLayerOverrides  
);constraintGroups
```

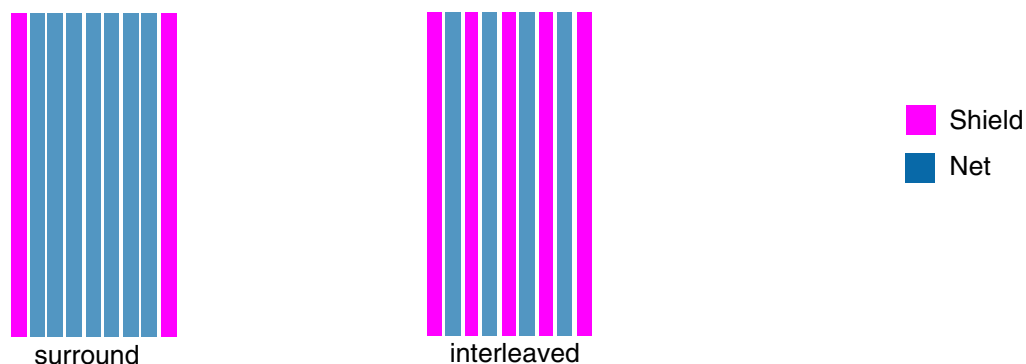
Applying the `myRoutingLayerOverrides` constraint group to a bounded area sets the following routing directions within that area:

- Metal1 to `rightDiag` instead of `horizontal`
- Metal2 to `horizontal` instead of `vertical`
- Metal3 to `none` instead of `rightDiag`; thereby, allowing it to take any routing direction, instead of being restricted to one

shieldGroupStyle

```
spacings(  
    ( shieldGroupStyle x_style )  
); spacings
```

Specifies the type of shielding for a group, such as one consisting of a diff pair or a bus constraint. Shielding can surround the group or interleave between bits.



Values

x_style

The shielding style for the group.

Valid values:

- **surround**: Shielding is done around the entire group. If the constraint is not specified, this value is considered by default.
- **interleaved**: A separate shield is used for each bit.

Parameters

None

Example

The required shielding type is `interleaved`.

```
spacings(  
    ( shieldGroupStyle "interleaved" )  
); spacings
```

shieldMinLength

```
spacings (
    ( shieldMinLength
        'shieldMinLengthValue x_shieldMinLengthValue
    )
) ;spacings
```

Specifies whether to shield the route segment if the length of the segment is greater than or equal to the specified length.

Parameters

'shieldMinLengthValue x_shieldMinLengthValue

When the value of this parameter is greater than 0, the route segments that have length greater than or equal to this value are shielded. If not specified or the value is less than or equal to 0, all route segments on the shield nets are shielded.

Example

The route segments that have length greater than or equal to 1.1 are shielded.

```
spacings (
    ( shieldMinLength
        'shieldMinLengthValue 1.1
    )
) ;spacings
```

shieldTerminal

```
spacings(  
    ( shieldTerminal )  
); spacings
```

Specifies whether all terminal shapes of the shielded net are shielded while adding shield wires.

strandNumber

```
spacings(  
    ( strandNumber tx_layer  
      x_strandNum  
    )  
);spacings
```

Defines the number of strands to use on the specified layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_strandNum</i>	The number of strands.

Parameters

None

Example

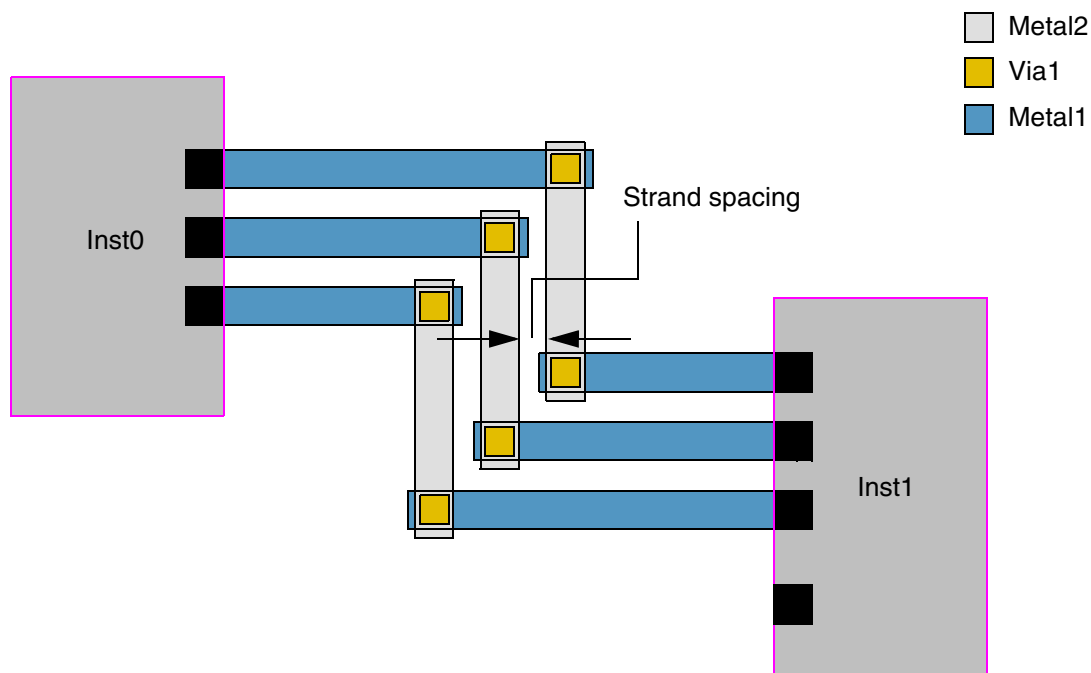
The number of strands that can be used on Metal1 is one.

```
spacings(  
    ( strandNumber "Metal1"  
      1  
    )  
);spacings
```


strandSpacing

```
spacings (
    ( strandSpacing tx_layer f_spacing )
) ;spacings
```

Specifies the spacing between individual wire strands.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between wire strands.

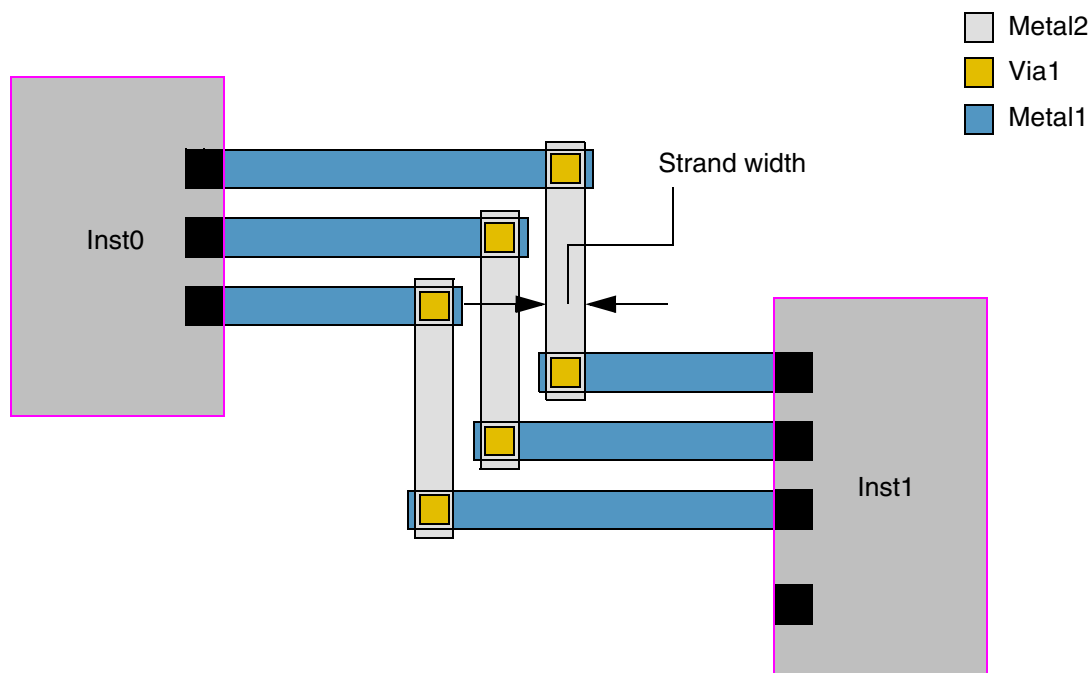
Parameters

None

strandWidth

```
spacings(  
    ( strandWidth tx_layer f_width )  
); spacings
```

Specifies the width of an individual wire strand.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_width</i>	The width of an individual wire strand.

Parameters

None

styleAroundVia

```
spacings (
    ( styleAroundVia x_style )
) ;spacings
```

Specifies the type of shielding around vias.

Use a value listed in the table below to specify the shielding type.

Name	Indicates
none	No shielding is used around vias.
around	Shield wires are used around vias.
allAround	Shield wires are used all around vias even in the wrong direction, if specified.
enclose	Shield wires are used all around vias in the preferred direction.

Values

x_style The shielding style for the via. Use a value listed in the table above to specify the shielding type.

Parameters

None

Example

The required shielding type is `around`.

```
spacings (
    ( styleAroundVia "around" )
) ;spacings
```

tieShield

```
spacings (
    ( tieShield
        ['shieldTieFreq
         ['shieldTieFreqValue x_shieldTieFreqValue]
        ]
        ['shieldRedundantVia]
    )
) ;spacings
```

Applies the constraint to vias based on whether or not ties are used for shield wires.

Parameters

'shieldTieFreq	Specifies that shields be tied at specified distances. Otherwise, by default, the shield engine will tie one connection per shield.
'shieldTieFreqValue <i>x_shieldTieFreqValue</i>	Specified when <i>shieldTieFreq</i> is used. The intervals at which connections are tied on shield wires. If a value is not specified or it is less than or equal to 0, the shield engine will tie only one connection per route segment.
'shieldRedundantVia	Specifies that new shield shapes will be connected with redundant or overlapping existing shield shapes during the tie shield operation. If not used, only one via is added per shield shape and no overlap connection will be made with existing shield shapes.

Example

Ties will be used to shield wires at a distance of 1.3. New shield shapes will be connected with redundant or overlapping existing shield shapes.

```
spacings (
    ( tieShield 'shieldTieFreq 'shieldTieFreqValue 1.3 'shieldRedundantVia)
) ;spacings
```

trackPattern

```
spacings(  
    ( trackPattern tx_layer t_trackPatternName )  
); spacings
```

Specifies track patterns for track-based routing.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t_trackPatternName</i>	The track pattern name. Multiple track pattern names can be specified by separating them with the "+" character. Type: String with the name of the track pattern

Parameters

None

Example

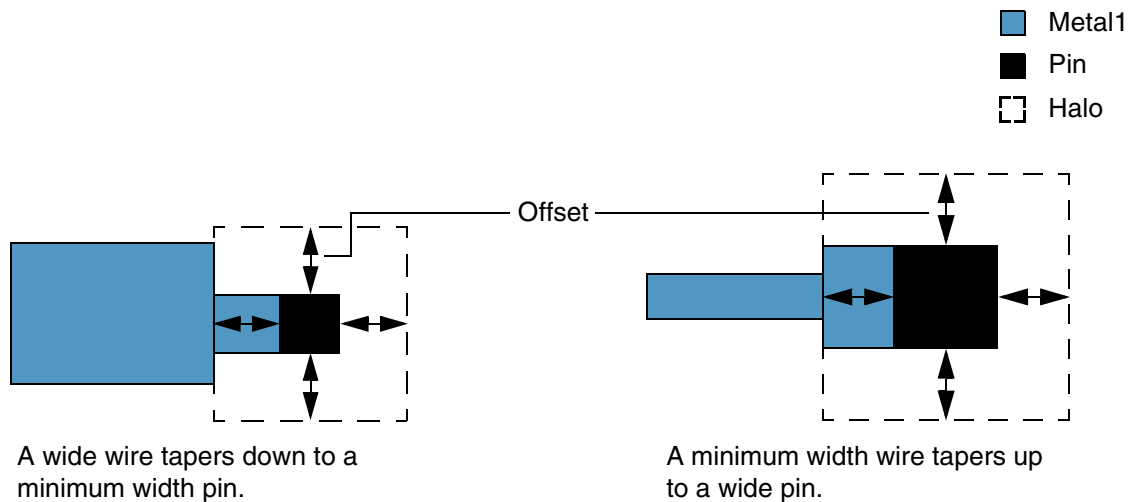
Adding the following constraint to a constraint group associated with a net restricts the routing of that net on Metal1 to tracks with track pattern "tp1".

```
spacings(  
    ( trackPattern "Metal1" "tp1" )  
); spacings
```

taperHalo

```
spacings(  
    ( taperHalo f_offset )  
); spacings
```

Establishes a window, or halo, around a pin in which the minimum width and minimum spacing constraints are applied when a wide wire is tapered down to a minimum width pin or when a minimum width wire tapers up to a wide pin.



Values

f_offset The offset for the taper halo.

Parameters

None

Example

The constraint establishes an offset of 1.5 for the halo used to taper wires to pins.

```
spacings(  
    ( taperHalo 1.5 )  
); spacings
```

validLayers

```
interconnect(  
    ( validLayers (tx_layer1 | (tx_layer1 tx_purpose1)  
                    [tx_layer2 | (tx_layer2 tx_purpose2)]  
                    ...  
                )  
    )  
);interconnect
```

Defines a list of valid routing layers and layer-purpose pairs.

Values

<i>tx_layer1</i>	The layer on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<i>tx_layer1 tx_purpose1</i>	The layer-purpose pair on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer and purpose numbers)

Parameters

None

Examples

Valid routing layers include Metal1, Metal2, and Metal3.

```
interconnect(  
    ( validLayers (Metal1 Metal2 Metal3) )  
);interconnect
```

Valid layers and layer-purpose pairs for routing include Metal1, (Metal2 pin), and Metal3.

```
interconnect(  
    ( validLayers (Metal1 (Metal2 pin) Metal3) )  
);interconnect
```

validPurposes

```
interconnect(  
    ( validPurposes  
        {'include | 'exclude}  
        (t_purpose ...)  
    )  
) ;interconnect
```

Specifies a list of purposes to be included or excluded.

All purposes to be included must be explicitly listed, including the `drawing` purpose. If a list of purposes to be excluded is specified, all purposes other than those explicitly listed are included.

Values

<i>t_purpose</i>	Purposes to be included or excluded. Type: A list of valid purposes, each enclosed in double quotes and separated by a space

Parameters

<code>'include 'exclude</code>	Include or exclude purposes. ■ <code>'include</code>: The listed purposes are included in the analysis by the target application. ■ <code>'exclude</code>: The listed purposes are excluded from analysis by the target application. Type: Boolean

Examples

- [Example 1: validPurposes with include](#)
- [Example 2: validPurposes with exclude](#)

Example 1: validPurposes with include

For layers M1, M2, and M3, only purposes "drawing", "vss", and "vdd" are used by the connectivity extractor.

```
( "virtuosoDefaultExtractorSetup" nil
  interconnect(
    ( validLayers ("M1" "M2" "M3") )
    ( validPurposes 'include ("drawing" "vdd" "vss") )
  ) ;interconnect
) ;virtuosoDefaultExtractorSetup
```

Example 2: validPurposes with exclude

For layers M1, M2, and M3, purpose "trackRegion" is excluded from use by the connectivity extractor.

```
( "virtuosoDefaultExtractorSetup" nil
  interconnect(
    ( validLayers ("M1" "M2" "M3") )
    ( validPurposes 'exclude ("trackRegion") )
  ) ;interconnect
) ;virtuosoDefaultExtractorSetup
```

validVias

```
interconnect(
    ( validVias ( t_viaName ... ))

    ( validVias
        (
            (( "width" nil nil "width" nil nil )
              'viaLayerDirection ((( "viaLayerDirection" )) (g_dirTable))
              l_default
            )
            (g_table)
        )
    )
) ;interconnect
```

Specifies a list of vias to be used to connect two wires, on two different layers, with width in a certain width range. Vias and via variants not listed by this constraint may not be recognized as valid routing vias. For more information about how to define vias and via variants, see [Via Definitions](#) in *Virtuoso® Technology Data ASCII Files Reference*.

Values

t_viaName The names of valid routing vias or via variants.

"width" nil nil "width" nil nil

This identifies the index for *table*.

g_table The via or via variant to be used to connect wires on different layers, depending on the width of the wires.

The table has the following format:

(*f_width1* *f_width2*) (*t_viaName* ...)

where,

- *f_width1* is the width of the wire on the bottom layer (*layer1*).
- *f_width2* is width of the wire on the top layer (*layer2*).
- *t_viaName* is the via or via variant that can be used to connect the two wires. Valid values include the name of a via or a via variant or *nil*.

Type: A 2-D table specifying width and via or via variant names

Parameters

<code>'viaLayerDirection</code>	A pair of direction values corresponding to each entry in <i>table</i>. The first direction value applies to <i>layer1</i> and the second direction applies <i>layer2</i> in the via definition. The table (<i>dirTable</i>) has the following format: <pre>x_index ((t_direction1 t_direction2) (t_direction1 t_direction2) ...)</pre> where, ■ <i>index</i> is a positive integer value starting at 0. ■ <i>direction1</i> and <i>direction2</i> can be horizontal or vertical. Type: A 1-D table specifying integer and string values
<code>l_default</code>	The vias to be used when no table entry applies.

Examples

- [Example 1: validVias](#)
- [Example 2: validVias \(2-D table\)](#)
- [Example 3: validVias with viaLayerDirection](#)

Example 1: validVias

```
interconnect(
  ( validVias ("via2" "via3" "via4" "via5" "via2Variant1") )
) ;interconnect
```

Example 2: validVias (2-D table)

```
interconnect(
  ( validVias
    (
      (( "width" nil nil "width" nil nil)
        (via2Variant1 via2)
      )
      (
        (0.6 0.7) (via2 via2Variant1)
        (0.8 0.9) (via3)
      )
    )
  )
) ;interconnect
```

Virtuoso Technology Data Constraints Reference

Routing Constraints

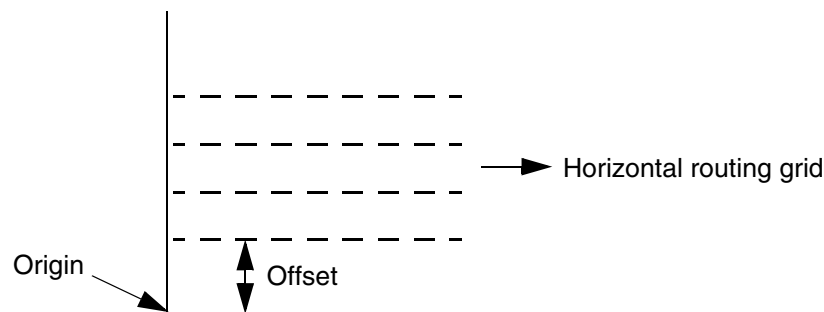
Example 3: validVias with viaLayerDirection

```
interconnect(
  ( validVias
    (
      (( "width" nil nil "width" nil nil)
        'viaLayerDirection ((( "viaLayerDirection" ))
          ( 0 (("horizontal" "vertical") ("horizontal" "vertical"))
            1 nil
            2 nil
            3 nil
          )
        )
      )
    (
      (0.72 4.5 ) (M2_M1_FP M3_M2_FP)
      (0.72 4.501) nil
      (0.721 4.5 ) nil
      (0.721 4.501) nil
    )
  )
) ;interconnect
```

verticalOffset

```
routingGrids(
    ( verticalOffset [tx_layer] f_offset )
    ...
) ;routingGrids
```

Sets the offset from the origin for the start of the horizontal routing grid. A vertical offset specified with a layer applies only to that layer. When specified without a layer, it sets the default for all layers not explicitly assigned a vertical offset.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_offset</i>	The offset from the origin. The default value is 0.

Parameters

None

Example

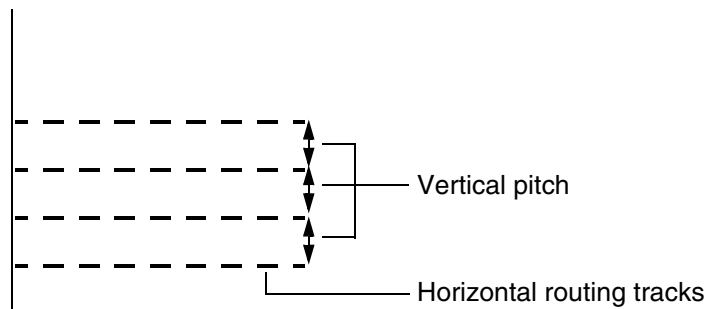
The vertical offset must be 1.2 for Metal3 and 1.4 for the rest of the layers.

```
routingGrids(
    ( verticalOffset "Metal3" 1.2 )
    ( verticalOffset 1.4 )
) ;routingGrids
```

verticalPitch (One layer)

```
routingGrids(
    ( verticalPitch [tx_layer]
      ['firstLastPitch f_firstLastPitch]
      f_pitch
    )
) ;routingGrids
```

Specifies the vertical pitch (distance) between the routing tracks on the specified layer. If *layer* is not specified, the constraint sets the default pitch for all layers for which a vertical pitch is not explicitly assigned.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_pitch</i>	The distance between the horizontal routing tracks on the specified layer. If 'firstLastPitch' is specified, this distance applies only to the tracks between the first and last routing tracks.

Parameters

'firstLastPitch *f_firstLastPitch*

The distance of the first and the last routing track from the cell row boundary.

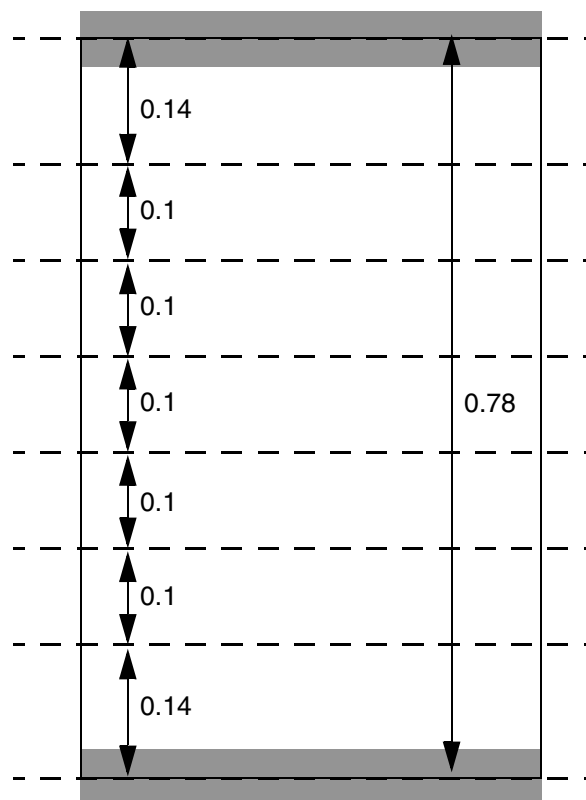
Virtuoso Technology Data Constraints Reference

Routing Constraints

Example

The vertical pitch is 0.14 for the first and last routing tracks and 0.1 for intermediate tracks. This applies to all layers for which vertical pitch is not explicitly assigned

```
routingGrids(  
  ( verticalPitch  
    'firstLastPitch 0.14  
    0.1  
  )  
) ;routingGrids
```



The dotted lines represent horizontal routing tracks. The first and last tracks are 0.14 away from the cell row boundary and the intermediate tracks are 0.1 apart. The cell row height is 0.78 ($2 \times 0.14 + 5 \times 0.1$).

voltageLayerMarkerMapping

```
spacings(  
    ( voltageLayerMarkerMapping tx_layer)  
        'layer t_layer  
        'purpose t_purpose  
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the LPP to be used for marking nets that transition in phase. The constraint applies on the layer on which the voltage is measured.

Values

<i>tx_layer</i>	The layer on which the voltage is measured. Type: String (layer and purpose names) or Integer (layer number)

Parameters

<i>'layer t_layer</i>	The layer to use for the marking shape.
<i>'purpose t_purpose</i>	The purpose to use for the marking shape.

Example

```
spacings(  
    ( voltageLayerMarkerMapping "Metall1"  
        'layer "Metall1"  
        'purpose "mlmin"  
    )  
) ;spacings
```


voltageLayerPairMarkerMapping

```
spacings (
    ( voltageLayerPairMarkerMapping tx_layer1 tx_layer2)
      'layer t_layer
      'purpose t_purpose
    ) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the LPP to be used for marking nets that transition in phase. The constraint applies on the two layers between which the voltage is measured.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

<i>'layer t_layer</i>	The layer to use for the marking shape.
<i>'purpose t_purpose</i>	The purpose to use for the marking shape.

Example

```
spacings (
    ( voltageLayerPairMarkerMapping "Metal1" "Metal2"
      'layer "Metal1"
      'purpose "mlmin"
    )
  ) ;spacings
```

Virtuoso Technology Data Constraints Reference

Routing Constraints

Spacing Constraints (One Layer)

This topic includes the following constraints:

- [allowedBetweenNeighborWidthRanges](#)
- [allowedSpacingRanges \(One layer\)](#)
- [bumpMinSpacing](#)
- [cornerEndOfLineKeepout](#)
- [cornerKeepOut](#)
- [cornerOnlyEndOfLineKeepout](#)
- [cornerFillSpacing](#)
- [cpodeGroup](#)
- [cpodeGroupSpacing](#)
- [deltaVLabelMapping](#)
- [deltaVLayerMarkerMapping](#)
- [deltaVLayerPairMarkerMapping](#)
- [deltaVoltage](#)
- [directionalEolKeepout](#)
- [endOfLineKeepout](#)
- [eolViaKeepout](#)
- [forbiddenEdgePitchRange](#)
- [forbiddenProximitySpacing](#)
- [forbiddenSpacingRanges](#)
- [forbiddenSpanLength](#)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- lineEndGap
- lithoMacroHalo
- maxFilling
- maxTapSpacing
- minAdjacentFourViaSpacing
- minCenterToCenterSpacing
- minClusterSpacing (One layer)
- minConcaveCornerSpacing
- minConvexCornerSpacing
- minCornerSpacing (One layer)
- minCornerToCornerDistance
- minCornerVoltageSpacing
- minDiagonalSpacing
- minDiffNetSpacing
- minEdgeLengthSpacing
- minEndOfLineExtensionSpacing
- minEndOfLinePerpSpacing
- minEndOfLineSpacing
- minEndOfLineToConcaveCornerSpacing
- minEndOfLineToNotchSpacing
- minEndOfNotchSpacing
- minExtensionSpacing
- minFillToFillSpacing
- minFillToShapeSpacing
- minFiveWiresEndOfLineSpacing
- minHoleWidth
- minInfluenceSpacing

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- minJointCornerSpacing
- minLargeNeighborViaArrayCutSpacing
- minLengthWithCut
- minNeighborVoltageSpacing (One layer)
- minNotchSpacing
- minNotchSpanSpacing
- minOppSpanSpacing
- minOrthogonalSpacing
- minOuterVertexSpacing
- minParallelSpanSpacing
- minPRBoundaryExteriorHalo
- minPRBoundaryInteriorHalo
- minPrITwoSidesSpacing
- minProtrusionSpacing
- minSameNetSpacing (One layer)
- minSideSpacing (One layer)
- minSpacing (One layer)
- minSpanLengthExtensionSpacing
- minSpanLengthSpacing
- minStubInfluenceSpacing
- minVoltageSpacing (One layer)
- oneSideSpacing
- onGridOnly
- pgViaTrack
- requiredEndOfLineShape
- shapeRequiredBetweenSpacing
- sixWiresSpacing

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- [snapPatternDefOffset](#)
- [threeShapesForbiddenSpacingRanges](#)
- [trimMinAdjacentSpacing](#)
- [trimMinSpacing \(One layer\)](#)
- [viaKeepoutZone](#)
- [voltageLabelMapping](#)
- [voltageMarkerMapping](#)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

allowedBetweenNeighborWidthRanges

```
spacings (
  ( allowedBetweenNeighborWidthRanges tx_layer
    (( "width" nil nil "width" nil nil )
      {'horizontal | 'vertical}
      'distanceWithin f_distanceWithin
      ['physicalMask]
      ['overrideWidths (f_overrideTable)]
      ['widthRanges (g_widthRanges)]
      ['sidewall f_sidewall]
    )
    (g_table)
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the allowed width ranges for a shape, using a 2-D table indexed by the width of the neighboring shapes.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil "width" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The allowed width ranges for a shape, indexed by the width of the neighboring shapes. Type: A 2-D table specifying floating-point width values and the allowed width <u>ranges</u>

Parameters

'horizontal 'vertical	The spacing applies in the specified direction. Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'distanceWithin f_distanceWithin`

The constraint applies only if the distance between the two neighboring shapes is less than or equal to this value.

`'physicalMask`

The constraint applies only if the neighboring shapes are on a physical mask. This parameter can be used only if the constraint is specified on a layer with two or more masks.

Type: Boolean

`'overrideWidths (f_overrideTable)`

Used to selectively override the constraint value for specific pairs of neighboring shape widths. The table is not interpolated. Instead, a lookup is performed only on the specified rows and columns.

Type: A 2-D table specifying floating-point width values

`'widthRanges (g_widthRanges)`

The constraint applies to neighboring shapes with widths in the specified range.

Type: Floating-point values specifying a range of widths

`'sidewall f_sidewall`

The distance to a neighboring shape must be equal to this value.

Examples

- Example 1: allowedBetweenNeighborWidthRanges with vertical, distanceWithin, and widthRanges
- Example 2: allowedBetweenNeighborWidthRanges with vertical, distanceWithin, overrideWidths, and widthRanges

Example 1: allowedBetweenNeighborWidthRanges with vertical, distanceWithin, and widthRanges

The constraint applies when the following conditions are met:

- The distance between the two outer shapes is 0.7 or less and the widths of the two neighboring shapes are in the list specified by `'widthRanges`.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- All widths are measured in the vertical direction.
- If both neighboring shapes have width less than 0.24, the width of the middle shape must be greater than or equal to 0.05 and less than or equal to 0.08.

```

spacings (
  ( allowedBetweenNeighborWidthRanges "Metal1"
    (( "width" nil nil "width" nil nil )
      'vertical
      'distanceWithin 0.7
      'widthRanges (0.2 0.22 0.24 0.26)
    )
  )
  (
    (0.20 0.20) ( "[0.05 0.08]" )
    (0.20 0.24) ( "[0.07 0.15]" )
    (0.24 0.20) ( "[0.07 0.15]" )
    (0.24 0.24) ( "[0.07 0.15]" )
  )
)
) ; spacings

```

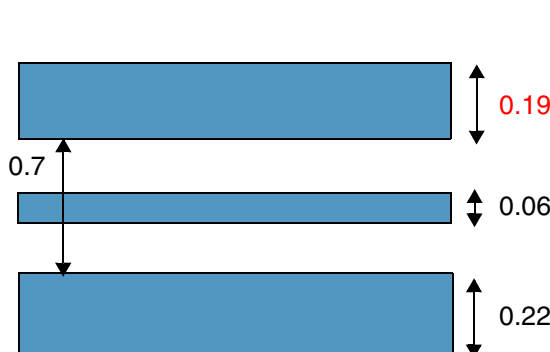
Metal1



a) PASS. The vertical distance between the two outer shapes is 0.7 (≤ 0.7). In addition, the shape between these two shapes has a width of 0.06, which falls in the allowed range, [0.05-0.08].



b) FAIL. The shape between the two outer shapes has a width of 0.09, which is not in the allowed range, [0.05-0.08].



c) The constraint does not apply because one of the neighboring shapes has a width other than 0.2, 0.22, 0.24, and 0.26.



d) The constraint does not apply because the neighboring shapes are at a distance greater than 0.7.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 2: *allowedBetweenNeighborWidthRanges* with *vertical*, *distanceWithin*, *overrideWidths*, and *widthRanges*

The constraint applies when the following conditions are met:

- The distance between the two outer shapes is 0.7 or less and the widths of the two neighboring shapes are in the list specified by 'widthRanges'.
- All widths are measured in the vertical direction.
- If both neighboring shapes have width less than 0.24, the width of the middle shape must be greater than or equal to 0.05 and less than or equal to 0.08.
- If the widths of the two outer shapes are 0.2 and 0.22, the width of the middle shape must be equal to 0.05 or greater than or equal to 0.07, but less than or equal to 0.08.

```

spacings (
  ( allowedBetweenNeighborWidthRanges "Metal1"
    (( "width" nil nil "width" nil nil )
      'vertical
      'distanceWithin 0.7
      'overrideWidths
      ((( "width1"      "width2" ))
        ( (0.20 0.22) (0.05 "[0.07 0.08]") ) )
      )
      'widthRanges (0.20 0.22 0.24 0.26)
    )
    (
      (0.20 0.20) ( "[0.05 0.08]" )
      (0.20 0.24) ( "[0.07 0.15]" )
      (0.24 0.20) ( "[0.07 0.15]" )
      (0.24 0.24) ( "[0.07 0.15]" )
    )
  )
) ;spacings

```

Metal1



a) FAIL. The widths of the neighboring shapes trigger the 'overrideWidths' parameter. However, the width of the middle shape is 0.06, which is not one of the allowed values, 0.05, [0.07-0.08].



b) PASS. The widths of the neighboring shapes do not trigger the 'overrideWidths' parameter. In addition, the width, 0.06, is within the allowed range, [0.05-0.08].

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

allowedSpacingRanges (One layer)

```
spacings (
  ( allowedSpacingRanges tx_layer
    ['sameMask | 'diffMask]
    ['mask1 | 'mask2 | 'mask3]
    ['exceptWidthEdge]
    ['width f_width]
    ['jointEOLWidthRanges g_jointEOLWidthRanges]
    ['horizontal | 'vertical]
    ['widthRanges (g_widthRanges)]
    ['exceptOverLayer tx_exceptOverLayer
      'overLayerPr1 f_pr1
      'overLayerWidthRanges (g_overLayerWidthRanges)
    ]
    ['exceptNumShapes g_exceptNumShapeRange
      'numShapeDistance f_distance
      'numShapeWidthRanges (g_numShapeWidthRanges)
      'numShapeMaxWidth f_maxWidth]
    ]
    ['stepSize f_stepSize ['stepRange g_stepRange]]
    ['overLayer tx_overLayer]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    (g_ranges)
  )
) ;spacings

spacingTables (
  ( allowedSpacingRanges tx_layer
    (( "width" nil nil ["width" nil nil] )
      ['sameMask | 'diffMask]
      ['mask1 | 'mask2 | 'mask3]
      ['exceptWidthEdge]
      ['width f_width]
      ['jointEOLWidthRanges g_jointEOLWidthRanges]
      ['parallelRunLengthTable]
      ['horizontal | 'vertical]
      ['widthRanges (g_widthRanges)]
      ['exceptOverLayer tx_exceptOverLayer
        'overLayerPr1 f_pr1
        'overLayerWidthRanges (g_overLayerWidthRanges)
      ]
      ['exceptNumShapes g_exceptNumShapeRange
        'numShapeDistance f_distance
        'numShapeWidthRanges (g_numShapeWidthRanges)
      ]
    )
  )
)
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
[ 'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
  [ 'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN) ]
  |
  'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
  [ 'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN) ]
]
[ f_default ]
)
(g_table)
)
) ;spacingTables
```

Specifies a set of allowed spacing ranges between two shapes on a layer. Spacing can be independent of the width of the shapes, dependent on the width of one shape, or dependent on the width of both shapes.

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.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The allowed spacing ranges specified as follows: (spacingRange1 spacingRange2 ...) Type: Floating-point values specifying a <u>range</u> of spacings that are allowed
"width" nil nil ["width" nil nil]	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

g_table

The format of a 1-D *table* row is as follows:

(f_width f_spacing)

where, *f_width* is the width of the wider of the two shapes and *f_spacing* is the spacing allowed when the width is greater than or equal to the corresponding index.

Note: If 'parallelRunLengthTable' is specified, the 1-D table is indexed on the parallel run length, instead of width.

The format of a 2-D table is as follows:

((f_width1 f_width2) f_spacing)

where, *f_width1* and *f_width2* represent the widths of the two shapes and *f_spacing* is the spacing allowed when both widths are greater than or equal to the corresponding indexes.

Type: A 1-D table specifying floating-point width (or parallel run length) and spacing values, or a 2-D table specifying floating-point width and spacing values

Parameters

'sameMask | 'diffMask

(Virtuoso Layout Suite EXL and higher tiers) The mask type.

- 'sameMask: The constraint applies only between shapes on the same mask.
- 'diffMask: The constraint applies only between shapes on different masks.

Type: Boolean

'mask1 | 'mask2 | 'mask3

(Virtuoso Layout Suite EXL and higher tiers) The mask of the shapes to which this rule belongs to.

Type: Boolean

'exceptWidthEdge

(Virtuoso Advanced Node for Layout Only) The constraint does not apply to any edges of a square shape or to edges whose span is considered as the width of the shape.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'width f_width`

The constraint applies only if the width of both adjacent shapes is less than this value.

`'jointEOLWidthRanges g_jointEOLWidthRanges`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only for L or T joints having end-of-line width in specified ranges.

`'horizontal | 'vertical`

The direction in which spacing is measured. If direction is not specified, spacing is measured in any direction.

Type: Boolean

`'widthRanges (g_widthRanges)`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to shapes with width in this range.

Type: Floating-point values specifying a range of widths that trigger the constraint

`'exceptOverLayer tx_exceptOverLayer 'overLayerPrl f_prl
'overLayerWidthRanges (g_overLayerWidthRanges)`

Note: These parameters together define an exemption that applies to two adjacent shapes.

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if all of the following conditions are met:

- The shapes overlap layer *exceptOverLayer*.
Type: String (layer name) or Integer (layer number)
- The parallel run length between the shapes is less than or equal to *prl*.
- The spacing between the shapes falls in the range specified by *overLayerWidthRanges*.

Type: Floating-point values specifying a range of spacings that are exempt

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
'exceptNumShapes g_exceptNumShapeRange 'numShapeDistance  
f_distance 'numShapeWidthRanges (g_numShapeWidthRanges)  
'numShapeMaxWidth f_maxWidth
```

Note: These parameters together define an exemption that applies to groups of adjacent shapes.

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if all of the following conditions are met:

- The number of shapes in a group is less than or equal to *exceptNumShapes*.

Type: Integer values specifying a range for the number of shapes that is exempt

- The distance between a pair of shapes in a group is less than or equal to *distance*.

- The spacing between two groups of shapes falls in the range specified by *numShapeWidthRanges*.

Type: Floating-point values specifying a range of spacings that are exempt

- The width of each shape in the group is less than or equal to *maxWidth*.

```
'stepSize f_stepSize
```

(Virtuoso Advanced Node for Layout Standard) The allowed spacings (S_{legal}) are restricted to the following values:

$$S_{\text{legal}} = S_{\text{min}} + n * \text{stepSize}$$

where, S_{min} is the lower end of the range specified by *stepRange* and $n \geq 0$. If *stepRange* is not specified, S_{min} is the lower end of the constraint value.

```
'stepRange g_stepRange
```

(Virtuoso Advanced Node for Layout Standard) The stepped spacing specified with *'stepSize* applies only in this range, which is fully contained in the constraint value.

Type: Floating-point values specifying a range of spacing

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'overLayer tx_overLayer`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to those shapes that overlap a single shape on this layer.

Type: String (layer name) or Integer (layer number)

`'parallelRunLengthTable`

The spacing table is indexed on the parallel run length between two shapes, instead of on width.

Type: Boolean

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`f_default`

The spacing value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Examples

- Example 1: allowedSpacingRanges with horizontal, exceptOverLayer, overLayerPrl, and overLayerWidthRanges
- Example 2: allowedSpacingRanges with horizontal, exceptNumShapes, numShapeDistance, and numShapeWidthRanges
- Example 3: allowedSpacingRanges with horizontal, exceptNumShapes, numShapeDistance, numShapeWidthRanges, and numShapeMaxWidth
- Example 4: allowedSpacingRanges with vertical and stepSize
- Example 5: allowedSpacingRanges with overLayer
- Example 6: allowedSpacingRanges with exceptWidthEdge
- Example 7: allowedSpacingRanges with sameMask and diffMask

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

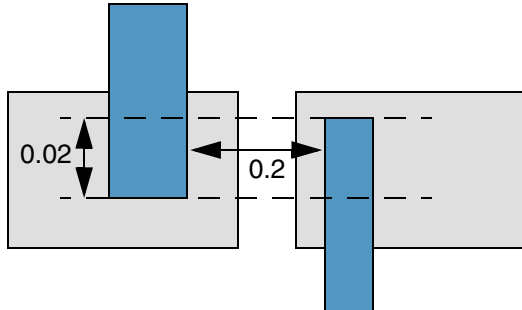
Example 1: *allowedSpacingRanges* with *horizontal*, *exceptOverLayer*, *overLayerPrl*, and *overLayerWidthRanges*

The horizontal spacing between two Metal1 shapes must be 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:

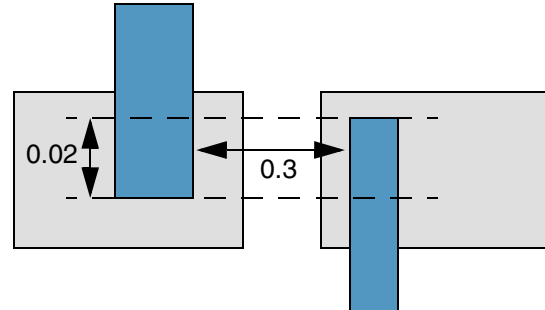
- The shapes overlap Metal2 shapes.
- The parallel run length between Metal1 shapes is less than or equal to 0.02.
- The spacing between Metal1 shapes is greater than or equal to 0.2 and less than or equal to 0.25.

```
spacings(  
  ( allowedSpacingRanges "Metal1"  
    'horizontal  
    'exceptOverLayer "Metal2"  
    'overLayerPrl 0.02  
    'overLayerWidthRanges ("[0.2 0.25]")  
    ("[0.1 0.19]" ">=0.4")  
  )  
) ;spacings
```

□ Metal2
■ Metal1



a) The constraint does not apply because all three exemption conditions are satisfied: Metal1 shapes overlap Metal2 shapes; parallel run length between Metal1 shapes is (less than or equal to) 0.02; and the spacing between Metal1 shapes is 0.2, which falls in the exempted range, [0.2 0.25].



b) FAIL. The constraint applies because the spacing between Metal1 shapes is 0.3, which lies outside the exempted range, [0.2 0.25], but fails because 0.3 does not fall in the allowed spacing ranges.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

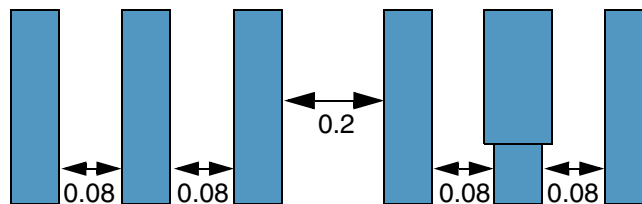
Example 2: *allowedSpacingRanges* with *horizontal*, *exceptNumShapes*, *numShapeDistance*, and *numShapeWidthRanges*

The horizontal spacing between two Metal1 shapes must be 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:

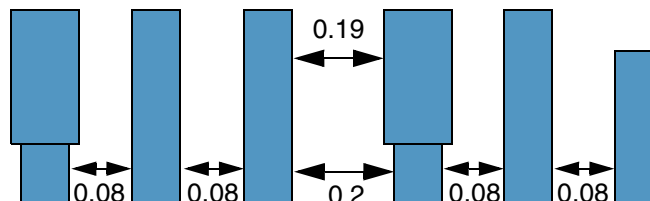
- The number of Metal1 shapes in a group is 3 or less.
- The distance between a pair of shapes in a group is less than or equal to 0.08.
- The spacing between two groups of shapes is greater than or equal to 0.15 and less than or equal to 0.2.

```
spacings(  
  ( allowedSpacingRanges "Metal1"  
    'horizontal  
    'exceptNumShapes 3  
    'numShapeDistance 0.08  
    'numShapeWidthRanges ("[0.15 0.2]")  
    ("[0.05 0.1]" ">=0.4")  
  )  
) ;spacings
```

■ Metal1



a) The constraint does not apply because all three exemption conditions are satisfied: There are three Metal1 shapes in each group; the distance between each pair of shapes in both groups is (less than or) equal to 0.08; and the spacing between the two groups is 0.2, which falls in the exempted range of [0.15 0.2].



b) FAIL. The exception does not apply because three shapes along the full parallel run length are required on each side.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

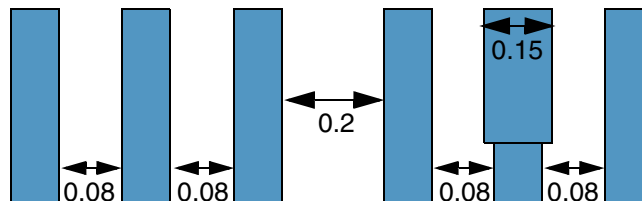
Example 3: *allowedSpacingRanges* with *horizontal*, *exceptNumShapes*, *numShapeDistance*, *numShapeWidthRanges*, and *numShapeMaxWidth*

The horizontal spacing between two Metal1 shapes must be 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:

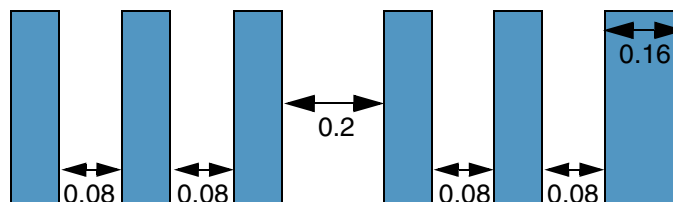
- The number of Metal1 shapes in a group is 3 or less.
- The distance between a pair of shapes in a group is less than or equal to 0.08.
- The spacing between two groups of shapes is greater than or equal to 0.15 and less than or equal to 0.2.
- The width of each shape in the group is less than or equal to 0.15.

```
spacings (  
  ( allowedSpacingRanges "Metal1"  
    'horizontal  
    'exceptNumShapes 3  
    'numShapeDistance 0.08  
    'numShapeWidthRanges ("[0.15 0.2]")  
    'numShapeMaxWidth 0.15  
    ("[0.05 0.1]" ">=0.4")  
  )  
) ;spacings
```

■ Metal1



a) The constraint does not apply because all four exemption conditions are satisfied: There are three Metal1 shapes in each group; the distance between each pair of shapes in both groups is (less than or) equal to 0.08; the spacing between the two groups is 0.2, which falls in the exempted range of [0.15 0.2]; and the width of the widest shape is (less than or) equal to 0.15.



b) FAIL. The constraint applies because at least one shape in the second group is 0.16 wide (and not less than or equal to 0.15), but fails because 0.2 does not fall in the allowed spacing ranges.

Virtuoso Technology Data Constraints Reference

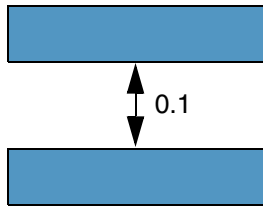
Spacing Constraints (One Layer)

Example 4: *allowedSpacingRanges* with vertical and stepSize

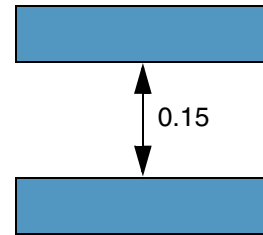
The vertical spacing between two Metal1 shapes must be greater than or equal to 0.1 and less than or equal to 0.2. A legal spacing value is the sum of the lower bound of this spacing range (0.1) and a multiple of 0.05.

```
spacings (  
  ( allowedSpacingRanges "Metal1"  
    'vertical  
    'stepSize 0.05  
    ("[0.1 0.2]")  
  )  
) ;spacings
```

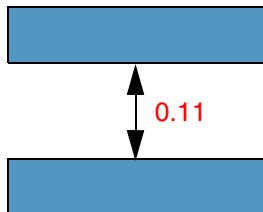
■ Metal1



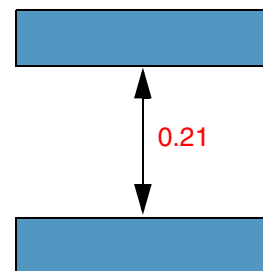
a) PASS. The spacing 0.1 falls in the allowed spacing range (when $n=0$, $S_{\text{legal}} = 0.1$ ($0.1+0 \times 0.05$)).



b) PASS. The spacing 0.15 falls in the allowed spacing range (when $n=1$, $S_{\text{legal}} = 0.15$ ($0.1+1 \times 0.05$)).



c) FAIL. The spacing 0.11 falls in the allowed spacing range, but is not allowed because it does not meet the stepped spacing requirement.



b) FAIL. The spacing 0.21 lies outside the allowed spacing range.

Virtuoso Technology Data Constraints Reference

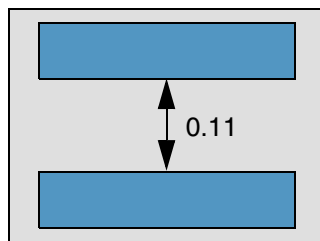
Spacing Constraints (One Layer)

Example 5: *allowedSpacingRanges* with *overLayer*

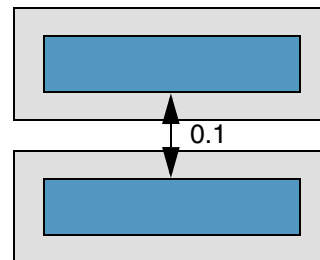
The spacing between two Metal1 shapes must be greater than 0.1 if both shapes overlap a single Metal2 shape.

```
spacings (  
  ( allowedSpacingRanges "Metal1"  
    'overLayer "Metal2"  
    (">0.1")  
  )  
) ;spacings
```

□ Metal2
■ Metal1



a) PASS. Both Metal1 shapes overlap a single Metal2 shape and the spacing between the two Metal1 shapes is 0.11 (>0.1).



b) The constraint does not apply because the two Metal1 shapes overlap different Metal2 shapes.

Virtuoso Technology Data Constraints Reference

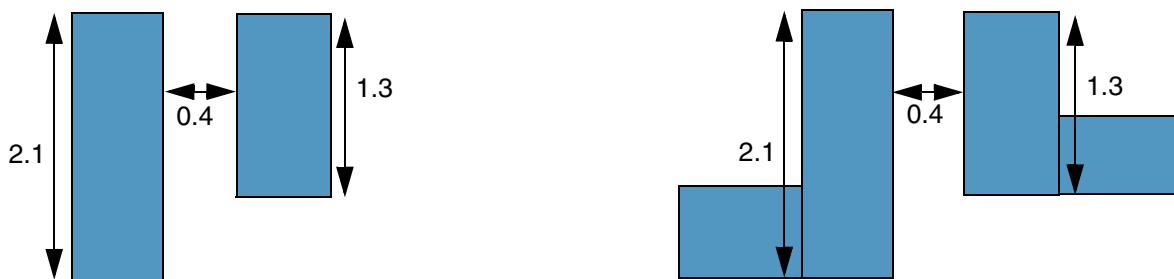
Spacing Constraints (One Layer)

Example 6: *allowedSpacingRanges* with *exceptWidthEdge*

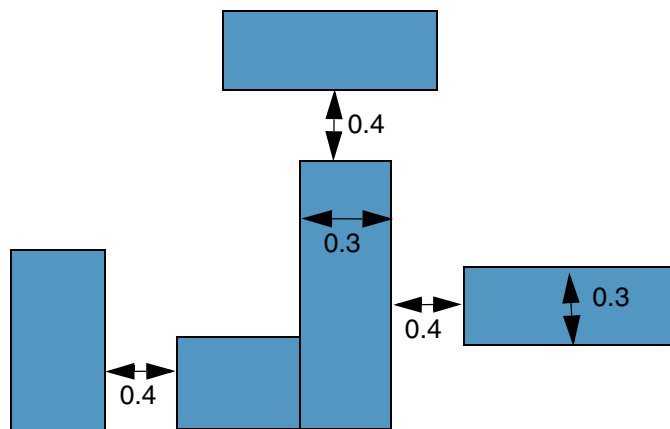
The spacing between two Metal1 shapes must be greater than 0.6 and parallel edges are length edges.

```
spacings (  
  ( allowedSpacingRanges "Metal1"  
    'exceptWidthEdge  
    (">0.6")  
  )  
) ;spacings
```

■ Metal1



FAIL. The constraint applies because both the parallel edges are length edges of the two shapes. It fails because the spacing is 0.4, which is less than 0.6.



PASS. The constraint does not apply because for each of the three spacing checks, at least one of the edges involved is a width edge.

Virtuoso Technology Data Constraints Reference

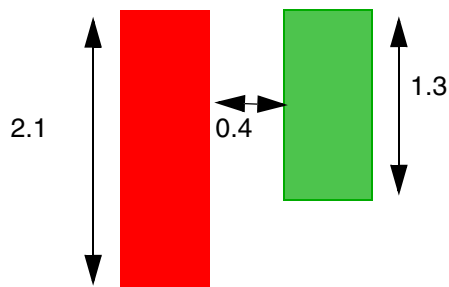
Spacing Constraints (One Layer)

Example 7: *allowedSpacingRanges* with *sameMask* and *diffMask*

The spacing between two Metal1 shapes must be greater than 0.6 and parallel edges are length edges.

```
spacings (  
  ( allowedSpacingRanges "M1"  
    'sameMask  
    'exceptWidthEdge  
    (">0.6")  
  )  
) ;spacings
```

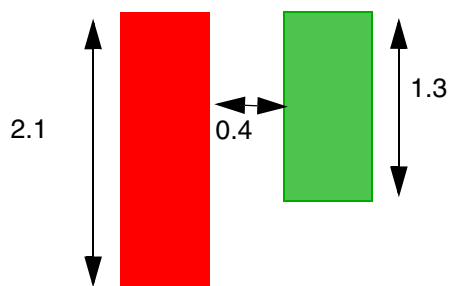
M1 (Mask 1)
 M1 (Mask 2)



PASS. The constraint does not apply because the shapes are not on the same mask.

```
spacings (  
  ( allowedSpacingRanges "M1"  
    'diffMask  
    'exceptWidthEdge  
    (">0.6")  
  )  
) ;spacings
```

M1 (Mask 1)
 M1 (Mask 2)



FAIL. The constraint applies, because the shapes are on different masks, and the constraint is violated because the spacing is only 0.4.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

bumpMinSpacing

```
spacings (
    ( bumpMinSpacing tx_layer
      'width f_width
      g_spacing
    )
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the bump to wire spacing if the width of the bump is greater than the bump width.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_spacing</i>	The minimum spacing between the bump and the wire on the specified layers.

Parameters

<i>'width f_width</i>	Specifies the width of a bump to trigger the bump spacing. The constraint applies when the bump width is greater than <i>f_width</i> .

Example

Specifies the bump to wire spacing.

```
spacings (
  (bumpMinSpacing "RDL1" 'width 10
  5.0)
); spacings
```

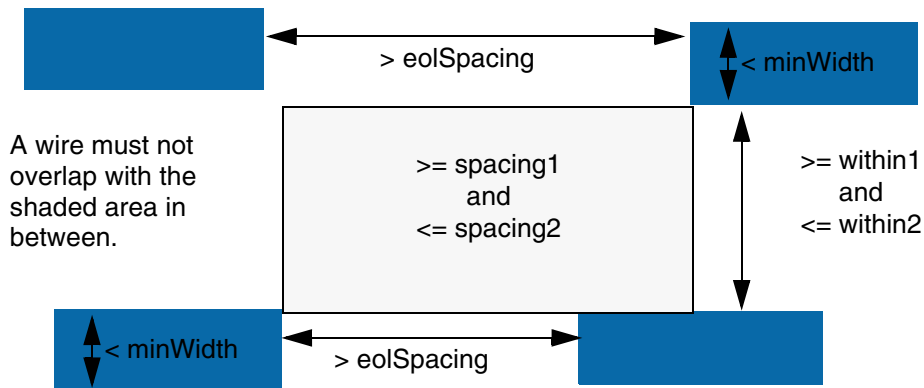
cornerEndOfLineKeepout

```

spacings (
  ( cornerEndOfLineKeepout tx_Layer
    'minWidth f_minWidth
    'eolSpacing g_eolSpacing
    'eolWithin ( g_within1 g_within2 )
    ( g_spacing1 g_spacing2 )
  )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Specifies that the spacing between two facing end-of-lines fulfilling certain given conditions cannot be within the given spacing range values, measured in the orthogonal direction of the end-of-lines.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>g_spacing1 g_spacing2</code>	The spacing range values, measured in the orthogonal direction of the end-of-lines. The spacing must not be greater than or equal to <code>g_spacing1</code> or less than or equal to <code>g_spacing2</code> . Type: Floating-point values specifying a <u>range</u> of spacings that are allowed

Parameters

<code>'minWidth f_minWidth</code>	The facing end-of-lines must have width less than the given value.
<code>'eolSpacing g_eolSpacing</code>	The spacing of the end-of-line to a neighbor wire with PRL greater than 0 to the end-of-line edge must be greater than the given value.
<code>'eolWithin (g_within1 g_within2)</code>	The facing end-of-lines must be within the given values (greater than or equal to the first given value and less than or equal to the second given value) in the direction of the end-of-lines.

Example

Specifies the spacing between two facing end-of-lines as follows:

- The facing end-of-lines must have width less than 0.05.
- The spacing of the end-of-line to a neighbor wire with PRL greater than 0 to the end-of-line edge must be greater than 0.1.
- The facing end-of-lines must be greater than or equal to the range 0.08 to 0.12 and less than or equal to the range 0.06 to 0.09 in the direction of the end-of-lines.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
spacings (  
  (cornerEndOfLineKeepout "Metal1"  
    'minWidth 0.05  
    'eolSpacing 0.1  
    'eolWithin (0.08 0.12)  
    (0.06 0.09))  
) ;spacings
```

cornerKeepOut

```
spacings(  
    ( cornerKeepOut tx_layer  
      ['includeAbutted']  
      (f_horizontal f_vertical)  
    )  
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies a rectangular keepout region at the corner of a shape. It is a violation if another corner overlaps this keepout region.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_horizontal f_vertical</i>	The dimensions of the keepout region in the vertical and horizontal directions.

Parameters

'includeAbutted	The constraint applies to shapes abutting the keepout window.
'otherWidthRanges	The constraint applies only if the width of the other wire is in the specified range.

Example

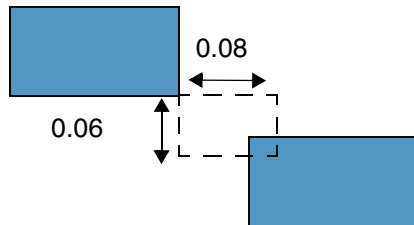
A keepout region of size 0.08 by 0.06 is created at each corner. It is a violation if a corner of another shape overlaps or touches this region.

Virtuoso Technology Data Constraints Reference

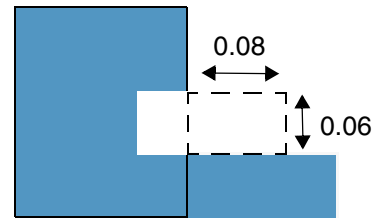
Spacing Constraints (One Layer)

```
spacings (
  (cornerKeepOut "Metal1"
    'includeAbutted
    (0.08 0.06)
  )
) ;spacings
```

■ Metal1



a) FAIL. A corner of another shape overlaps the corner keepout window.



b) PASS. No other corner is overlapping the keepout window. With the 'includeAbutted' option also the specified constraint passes because only abutting corners cause a violation.

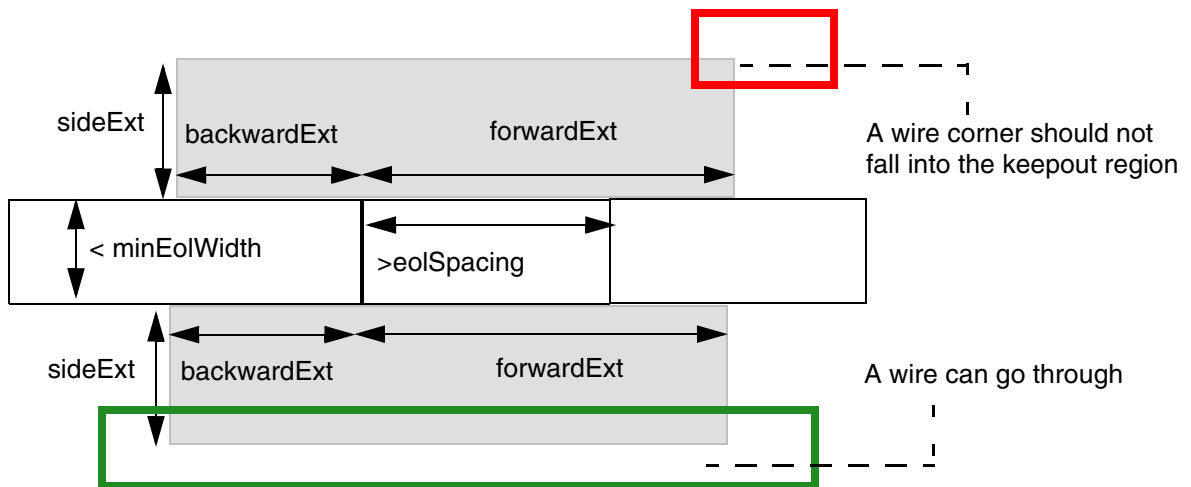
cornerOnlyEndOfLineKeepout

```

spacings (
  ( cornerOnlyEndOfLineKeepout tx_Layer
    'minEolWidth g_minEolWidth
    'eolSpacing g_eolSpacing
    ( f_backwardExt f_sideExt f_forwardExt )
  )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Specifies a keepout region for a neighbor wire corner formed by three given values by extending from the corners of an end-of-line edge that fulfills certain given conditions.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

tx_layer The layer on which the constraint is applied.
Type: String (layer and purpose names) or Integer (layer number)

f_backwardExt f_sideExt f_forwardExt

The amount of extension from the corner of the end-of-line edge:

- *f_backwardExt* indicates extension in the backward direction.
- *f_sideExt* indicates extension to the side.
- *f_forwardExt* indicates extension in the forward direction.

Parameters

'minEolWidth g_minEolWidth

The constraint applies when the facing end-of-line edge has width less than the given value.

'eolSpacing g_eolSpacing

The constraint applies when the spacing of the end-of-line edge to a neighbor wire with PRL greater than 0 to the end-of-line edge is greater than the given value.

Example

Specifies a keepout region for a neighbor wire corner when the following conditions are met:

- The facing end-of-line edge has width less than 0.05.
- The spacing of the end-of-line edge to a neighbor wire with PRL greater than 0 to the end-of-line edge must be greater than 0.1.

The facing end-of-line edges have a keepout area extending by 0.08, 0.06, and 0.09 in the backward, side, and forward directions, respectively.

```
spacings (  
    (cornerOnlyEndOfLineKeepout "Metal1"  
        'eolWidth 0.05
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
'eolSpacing 0.1  
(0.08 0.06 0.09)  
) ;spacings
```

cornerFillSpacing

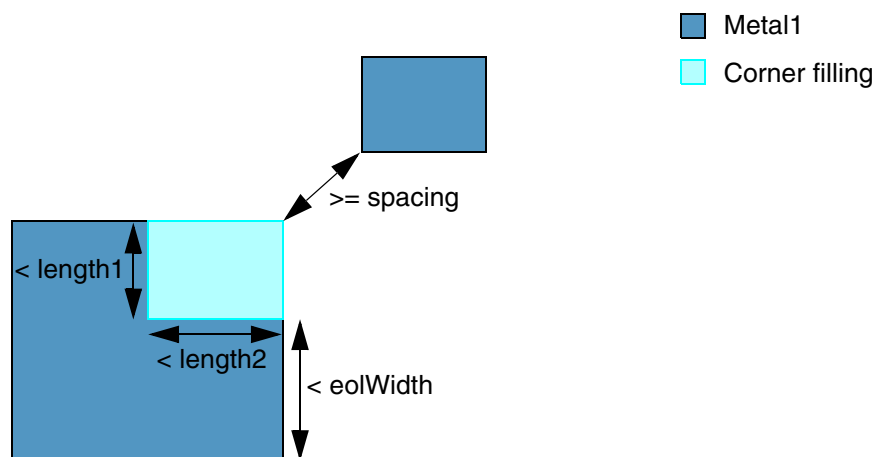
```
spacings (  
    ( cornerFillSpacing tx_layer  
      'width f_eolwidth  
      'cornerLengths (f_length1 f_length2)  
      f_spacing  
    )  
) ;spacings
```

Specifies the minimum spacing required between a convex corner that is formed by filling up a concave corner and a corner of a neighboring shape on a layer.

The concave corner is filled up before mask fabrication, so other shapes on that layer are far enough away from the convex corner that is formed to prevent spacing violations.

The constraint applies only if the following two conditions are satisfied:

- The lengths of the two edges that form the concave corner must be less than *length1* and *length2*.
- The edge with length less than *length2* must have an adjacent end-of-line edge with width less than *eolWidth*.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing must be greater than or equal to this value.

Parameters

<code>'width f_eolwidth</code>	The edge of the concave corner with length <i>length2</i> must have an adjoining end-of-line edge with width less than <i>eolWidth</i> .
<code>'cornerLengths (f_length1 f_length2)</code>	The constraint applies if the length of one edge of the concave corner is less than <i>length1</i> and the length of the other edge is less than <i>length2</i> . The edge with length <i>length2</i> must have an adjoining end-of-line edge with width less than <i>eolWidth</i> .

Example

The spacing between the missing convex corner of a shape and a corner of a neighboring shape must be at least 0.05 if the following conditions are met:

- The length of one concave side is less than 0.06 and the length of the other concave side is less than 0.08.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

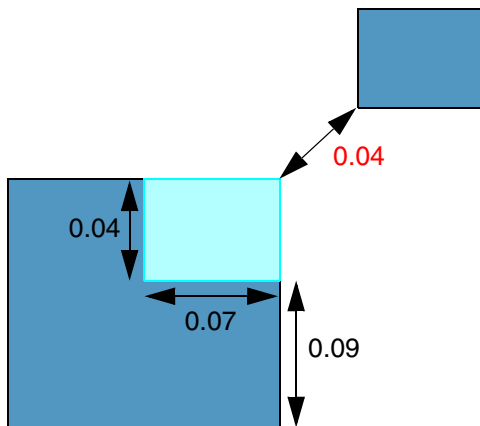
- The edge adjoining the concave edge with length less than 0.08 is an end-of-line edge with width less than 0.1.

```

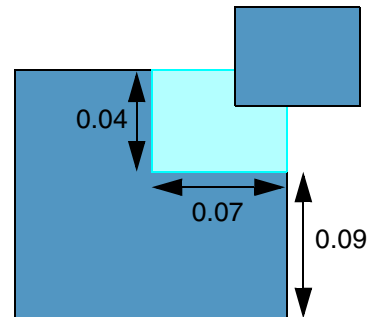
spacings(
  ( cornerFillSpacing Metall
    'width 0.1
    'cornerLengths (0.06 0.08)
    0.05
  )
) ;spacings

```

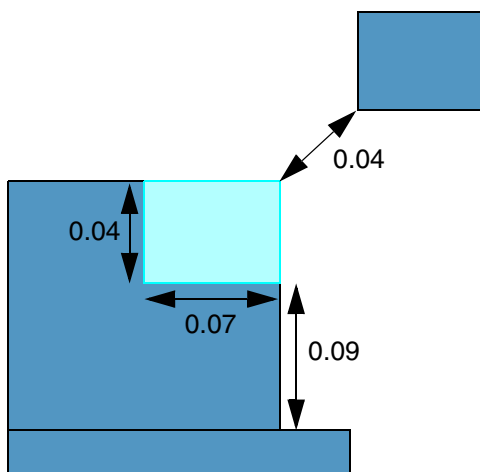
■ Metal1
■ Corner filling



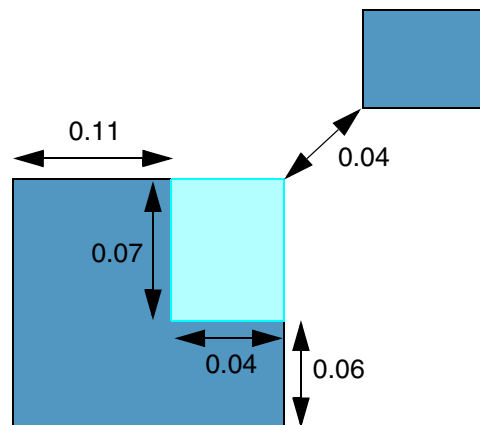
a) FAIL. Both required conditions are met, but the spacing is only 0.04 (<0.05).



b) FAIL. Both required conditions are met, but the neighboring shape overlaps the corner filling.



c) The constraint does not apply because the edge with width 0.09 is not an end-of-line edge.



d) The constraint does not apply because the end-of-line edge adjoining the concave edge with length 0.07 is 0.11 wide (>0.1).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Group

```
spacings (
    ( ccodeGroup tx_layer
      'prl g_parallelRunLength
      'parallelWithin g_parallelWithin
      g_ccodeNum
    )
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies that maximum *g_ccodeNum* CCODE shapes with PRL greater than or equal to the value specified in the *g_parallelRunLength* parameter and within the *g_parallelWithin* parameter forms a CCODE group.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ccodeNum</i>	The maximum number of shapes that form a CCODE group. If zero is specified, two or more shapes form a group.

Parameters

'prl <i>g_parallelRunLength</i>	Specifies that CCODE shapes with PRL greater than or equal to the specified value and within the value in <i>parallelWithin</i> can form a group.
'parallelWithin <i>g_parallelWithin</i>	Specifies that CCODE shapes with PRL greater than or equal to the specified value in <i>parallelRunLength</i> and within the specified value can form a group.

Example

```
spacings (
    (ccodeGroup "Ccode" 'prl 0.5 'parallelWithin 0.07 4)
); spacings
```


cpodeGroupSpacing

```
spacings(  
    ( cpodeGroupSpacing tx_layer  
      'numCpode g_numCpode  
      g_spacing  
    )  
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the required spacing between two CPODE groups with at least one of the groups having its number of shapes greater than or equal to the value specified in the *g_numCpode* parameter.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_spacing</i>	The minimum spacing between two CPODE groups.

Parameters

'numCpode *g_numCpode*

Specifies that the spacing is applied on the specified CPODE group with number of CPODE greater than or equal to the specified value.

Example

```
spacings(  
    (cpodeGroupSpacing "Cpode" 'numCpode 2  
      0.08)  
) ;spacings
```

deltaVLabelMapping

```
spacings (
    ( deltaVLabelMapping tx_layer
      'layer tx_labelLayer
      'purpose tx_labelPurpose
    )
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the layer and purpose of the `deltaV` label generated to represent the maximum voltage delta between two shapes on different nets. The `deltaV` label overlaps the `deltaV` marker shape on the constraints layer generated by using the `deltaVLayerMarkerMapping` or `deltaVLayerPairMarkerMapping` constraints. This marker overlaps the two net shapes.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Value

tx_layer The layer on which the `deltaV` marker is generated.
Type: String (layer and purpose names) or Integer (layer number)

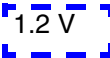
Parameters

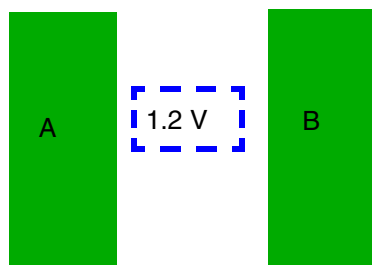
`'layer tx_labelLayer`
 The layer on which the `deltaV` label is generated.
Type: String (layer and purpose names) or Integer (layer number)

`'purpose tx_labelPurpose`
 The purpose on which the `deltaV` label is generated.
Type: String (purpose name)

Example

```
spacings(  
  ( deltaVLabelMapping "PO"  
    'layer "PO"  
    'purpose "userDV"  
  )  
) ;spacings
```

 PO drawing shape
 PO userDV label



deltaVLayerMarkerMapping

```
spacings(  
    ( deltaVLayerMarkerMapping tx_layer  
      'layer tx_markerLayer  
      'purpose tx_markerPurpose  
    )  
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) This constraint is used to indicate that a marker shape is generated to overlap two shapes on different nets on the constraint layer to record the maximum delta voltage between them. The constraint specifies the layer and purpose of the marker shape generated.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Value

tx_layer The layer on which the two different net shapes are found.
Type: String (layer and purpose names) or Integer (layer number)

Parameters

'layer tx_markerLayer The layer on which the marker shape overlapping the two net shapes are to be generated.
Type: String (layer and purpose names) or Integer (layer number)

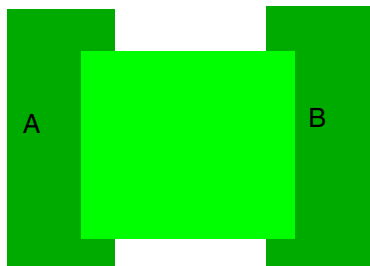
'purpose tx_markerPurpose The purpose on which the marker shape overlapping the two net shapes are to be generated.
Type: String (purpose name)

Example

```
spacings(  
  ( deltaVLayerMarkerMapping "PO"  
    'layer "PO"  
    'purpose "userDV_m"  
  )  
) ;spacings
```

 PO drawing shape

 PO userDV_m marker



deltaVLayerPairMarkerMapping

```
spacings(  
    ( deltaVLayerPairMarkerMapping tx_layer1 tx_layer2  
      'layer tx_markerLayer  
      'purpose tx_markerPurpose  
    )  
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) This constraint is used to indicate that a marker shape is generated to overlap two shapes, one on each layer of the constraint. These shapes are on different nets. The marker shape is used to record the maximum delta voltage between them. The constraint specifies the layer and purpose of the marker shape generated.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer1</i>	The layer on which one of the two net shapes is found. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The layer on which the other of the two net shapes is found. Type: String (layer and purpose names) or Integer (layer number)

Parameters

<i>'layer tx_markerLayer</i>	The layer on which the marker shape overlapping the two net shapes is generated. Type: String (layer and purpose names) or Integer (layer number)
<i>'purpose tx_markerPurpose</i>	The purpose on which the marker shape overlapping the two net shapes is generated. Type: String (purpose name)

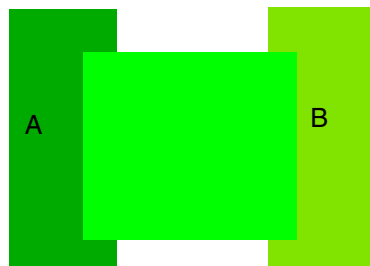
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

```
spacings (
  ( deltaVLayerPairMarkerMapping "MD" "PO"
    'layer "MD"
    'purpose "userDV_m"
  )
) ;spacings
```

-  PO drawing shape
-  MD drawing shape
-  PO userDV_m marker shape



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

deltaVoltage

```
spacings(  
    ( deltaVoltage f_voltage  
    )  
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the delta voltage between two nets.

Values

<i>f_voltage</i>	The constraint applies only if the maximum voltage difference between the two shapes is greater than this value.

Parameters

None

directionalEolKeepout

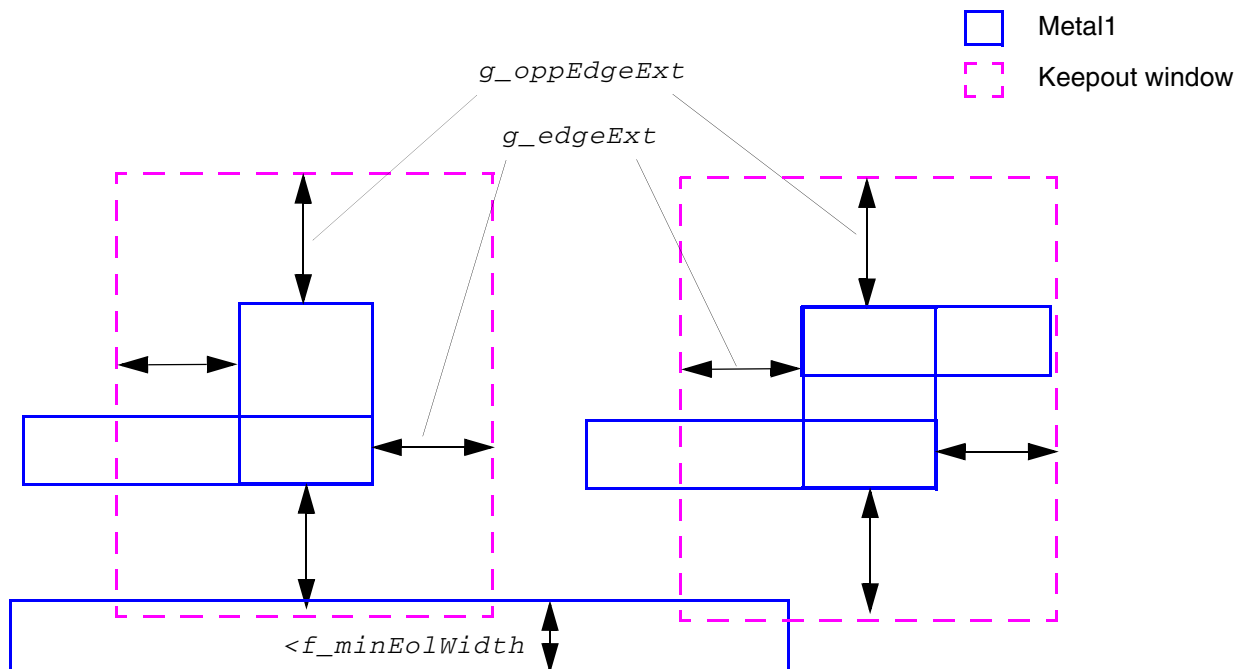
```

spacings (
    ( directionalEolKeepout tx_layer
      {'horizontal' | 'vertical'}
      'edgeLength' (f_minEdgeLength f_maxEdgeLength)
      'width' f_width
      'minEolWidth' f_minEolWidth
      (g_edgeExt g_oppositeEdgeExt)
    )
) ;spacings

```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the extension of a wire of a specific width in a specific direction. The resulting window must not overlap with any end-of-line.

A wire with width equal to f_width measured in the orthogonal direction would form a window by extending by $g\_edgeExt$ and by $g\_oppositeEdgeExt$ in the specified direction. The wire must have an edge in the specified direction that is more than $f\_minEdgeLength$ and less than $f\_maxEdgeLength$. The window must not overlap with any end-of-line with width less than $f\_minEolWidth$.



Assuming that the length of the edges is within the range specified by `edgeLength`:

- The pattern on the left passes the constraint. The bottom neighbor wire is a through wire without an EOL.
- The pattern on the right has two violations. The bottom wire has an EOL overlapping with the dotted window. In addition, the right EOL in the Z shape is also a violation because it falls inside the window.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_edgeExt g_oppEdgeExt</i>	Specifies the extensions for the window. A window is formed around a wire by extending <i>g_edgeExt</i> in the orthogonal direction and <i>g_oppEdgeExt</i> in the specified direction. Type: Floating-point values specifying a <u>range</u> of spacings that are allowed

Parameters

<i>'horizontal 'vertical</i>	Specifies the direction in which <i>edgeLength</i> is measured.
<i>'edgeLength (f_minEdgeLength f_maxEdgeLength)</i>	Specifies that the edge length must be greater than <i>f_minEdgeLength</i> and less than <i>f_maxEdgeLength</i> .
<i>'width f_width</i>	Specifies the exact width of a wire in the direction orthogonal to the specified direction.
<i>'minEolWidth f_minEolWidth</i>	Specifies that the window must not overlap with any end-of-line with width less than <i>f_minEolWidth</i> .

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

Specifies that a wire with width 0.05 in the preferred routing direction would form a window by extending 0.06 in the preferred routing direction and 0.08 in the non-preferred routing direction. The wire must have an edge in the non-preferred direction that is more than 0.04 and less than 0.09. The window must not overlap with any end-of-line with width less than 0.07.

```
spacings (
  (directionalEolKeepout "Metall"
    'vertical
    'edgeLength (0.04 0.09)
    'width 0.05
    'minEolWidth 0.07
    (0.06 0.08))
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

endOfLineKeepout

```
spacings (
    ( endOfLineKeepout tx_layer
        'width f_width
        ['horizontal | 'vertical]
        ['minEolWidth f_minEolWidth]
        ['widthRanges (g_widthRanges)]
        ['cornerOnly
            [{ 'exceptFromFrontEdge
                | 'exceptFromBackEdge
                ['forwardGap f_forwardGap]
                ['exactAligned f_exactAligned]
                ['extraForwardKeepout
                    (f_minKeepout f_maxKeepout)]
            }
            ['otherEol ['dualEolSpacing f_minEolWidth f_maxEolWidth]]]
            ['twoSides]
        ]
        ['exceptSpacing t_spacing | 'exceptLineEndGap]
        ['withinCut t_withinCut
            'cutLayers l_cutLayers]
        ['exceptEolSpacing t_eolSpacing]
    ]
    ['diffMask | 'sameMask]
    ['exceptSameMetal] ['exceptSideWithin f_withinRange]
    ['orGroupID x_orGroupID ['otherEndEol]]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['exceptSameSideMetal (t_backwardExt1 t_sideExt1 t_forwardExt1)]
    ['exceptNoOppSideMetal (t_backwardExt2 t_sideExt2 t_forwardExt2)]
    ['eolSpacingRange g_eolSpacingRange
        ['eolSideExtensionRange f_eolSideExtensionRange]]
    ['otherEolSpacingRange g_otherEolSpacingRange]
    ['jointAsEol]
    ['checkOtherWidth] | ['otherWidthRanges g_otherWidthRanges]
    ['equalRectWidth]
    ['ignoreJoint]
    ['orthogonalPair]
    ['mask1 | 'mask2 | 'mask3]
    (f_backwardExt f_sideExt f_forwardExt)
)
) ;spacings
```

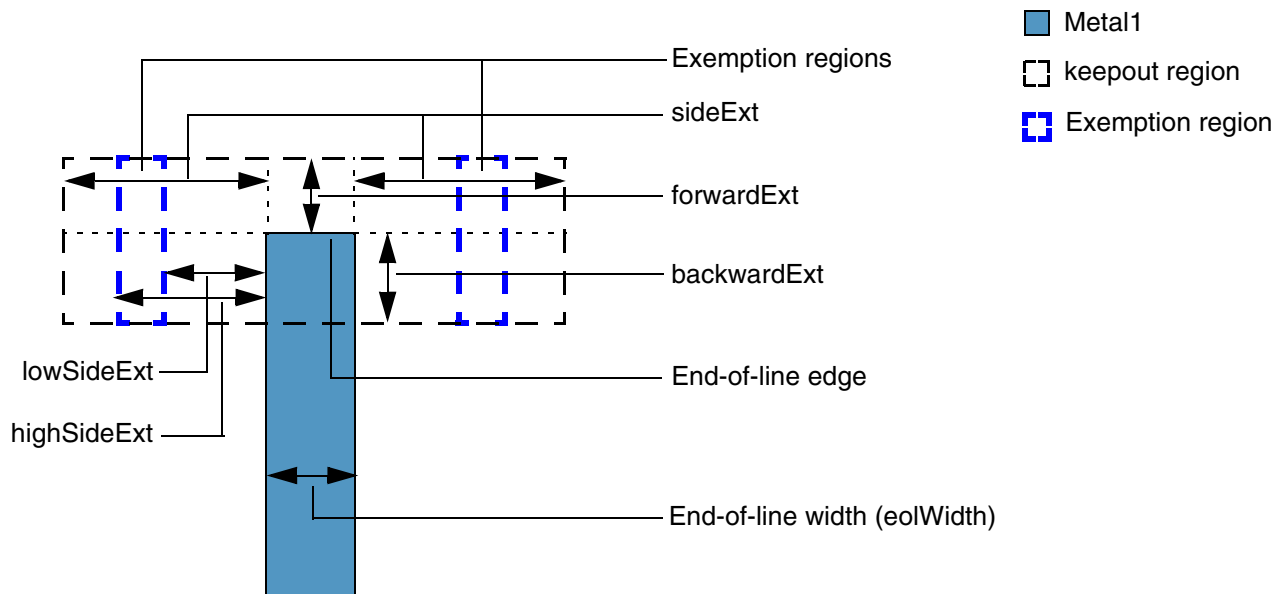
Specifies the keepout region around the end-of-line edge for shapes with end-of-line width less than the given value. Shapes on the same layer must not intersect this keepout region,

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

which extends both forward (*forwardExt*) and backward (*backwardExt*) from the end-of-line edge and on both sides (*sideExt*) of the end-of-line edge.

In some processes, a violation occurs if a corner of another shape intersects the keepout region, while in other processes, neither a corner nor an edge of another shape can intersect the keepout region. However, in some processes, an intersecting corner or edge of a contiguous same-metal shape is not considered a violation.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_backwardExt</i>	The backward extension value of the keepout region.
<i>f_sideExt</i>	The side extension value of the keepout region.
<i>f_forwardExt</i>	The forward extension value of the keepout region.

Parameters

<i>'width f_width</i>	The constraint applies to shapes with end-of-line width less than this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'horizontal | 'vertical

(Virtuoso Layout Suite EXL and higher tiers) The direction in which the constraint is applied to ends-of-lines. If it is not specified, the constraint is applied to all ends-of-lines.

Type: Boolean

'minEolWidth *f_minEolWidth*

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the end-of-line width is greater than or equal to the given value.

'widthRanges *g_widthRanges*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the end-of-line width falls in these width ranges and also meets *f_width*.

Type: Floating-point numbers specifying width ranges

'cornerOnly

Only a corner inside the keepout region is a violation. Otherwise, an edge or a corner of a neighboring shape intersecting the keepout region is a violation.

Type: Boolean

'exceptSameMetal

The corner or edge of a contiguous same-metal shape intersecting the keepout region is not a violation.

Type: Boolean

'exceptFromFrontEdge

(Virtuoso Advanced Node for Layout Standard) Neighboring shapes that intersect the front (top) edge of the keepout region do not cause a violation.

Type: Boolean

'exceptFromBackEdge

(Virtuoso Advanced Node for Layout Standard) Neighboring shapes that intersect the back (bottom) edge of the keepout region do not cause a violation.

Type: Boolean

'forwardGap *f_forwardGap*

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Advanced Node for Layout Only) If the spacing between the shape that is face-to-face with the end-of-line shape (inside the search window defined by *backwardExt*, *sideExt*, and *forwardGap*) is less than the *forwardGap* value, then it must be equal to the *forwardExt* value specified for the constraint.

`'exactAligned f_exactAligned`

(Virtuoso Layout Suite EXL and higher tiers) A different spacing or search window extension between two aligned end-of-line edges with a same width to be the specified value.

`'extraForwardKeepout (f_minKeepout f_maxKeepout)`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that an extra keepout is formed by the side extension and the specified values in the forward extension.

`'otherEol`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if the neighbor wire should also be an end-of-line.

`'dualEolSpacing f_minEolWidth f_maxEolWidth`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if the neighbor end-of-line has a width greater than or equal to the *f_minEolWidth* value and is less than the *f_maxEolWidth* value.

`'mask1 | 'mask2 | 'mask3`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the end-of-line shape is on the specified mask.

Type: Boolean

`'twoSides`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the end-of-line shape has two neighboring shapes, one on each side, in the keepout region.

Type: Boolean

`'diffMask | 'samMask`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Layout Suite EXL and higher tiers) The mask type.

- 'diffMask: The constraint applies only between shapes on different masks.
- 'sameMask: The constraint applies only between shapes on the same mask.

Type: Boolean

'exceptSpacing *t_spacing*

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply when the line-end spacing of the neighbor wire is equal to this given value.

'exceptLineEndGap

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply if the lineEndGap constraint condition on this layer is met. This exemption would be applied on a triggering line end or as a neighbor wire meeting the `lineEndGap` constraint condition.

'withinCut *t_withinCut* 'cutLayers *l_cutLayers*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies when there is a cut on one of the layers specified with *l_cutLayers* within the given value of *t_withinCut* to the line-end. A maximum of two cut layers can be specified with *l_cutLayers*.

'exceptEolSpacing *t_eolSpacing*

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply when the spacing of the line-end edge is less than or equal to this value.

'exceptSideWithin *f_withinRange*

(Virtuoso Advanced Node for Layout Standard) The extension values (*lowSideExt* and *highSideExt*) that define the exemption regions on both sides of the end-of-line edge. Shapes that intersect these exemption regions do not cause a violation.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'orGroupID x_orGroupID`

(Virtuoso Advanced Node for Layout Standard) An ID of type integer. All constraints with the same ID are treated as OR constraints; constraints with a different ID are treated as AND constraints. Typically, `'orGroupID` is used with one constraint value for an `'exceptFromBackEdge` region and with another constraint value for an `'exceptFromFrontEdge` region. In such cases, it is a violation if both constraints fail on the same side of the end-of-line shape.

`'otherEndEol`

(Virtuoso Advanced Node for Layout Only) If the edge found inside the keepout region is an end-of-line edge, a violation occurs only if this "other" end-of-line edge also fails all `endOfLineKeepout` constraints in an OR or AND group. This parameter must be specified along with `'exceptFromBackEdge` to ensure symmetry, which means that either end-of-line edge as a trigger edge can find the other end-of-line edge.

Type: Boolean

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
| `'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'exceptSameSideMetal (t_backwardExt1 t_sideExt1 t_forwardExt1)`

(Virtuoso Layout Suite EXL and higher tiers) Specifies a search window formed by three values defining the backward extension, side extension, and forward extension based on the line-end. If there is another wire in that window on the same side of a wire triggering the violation, the constraint does not apply.

`'exceptNoOppSideMetal (t_backwardExt2 t_sideExt2 t_forwardExt2)`

(Virtuoso Layout Suite EXL and higher tiers) Specifies a search window formed by three values defining the backward extension, side extension, and forward extension based on the line-end. If there is no wire in the window on the opposite side of a wire triggering the violation, the violation is exempted.

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'eolSpacingRange g_eolSpacingRange`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies when the spacing of the line-end edge of the triggering EOL is in this range.

`'eolSideExtensionRange f_eolSideExtensionRange`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies when no shapes are present in the window formed with the EOL spacing and the value specified by this parameter.

`'jointAsEol`

(Virtuoso Layout Suite EXL and higher tiers) The two edges at the joint of two intersecting shapes are also treated as end-of-line. For the edge to be considered end-of-line, one of the edges of the intersecting shapes that forms the joint edge must have width less than the width parameter.

`'checkOtherWidth | 'otherWidthRanges (g_otherWidthRanges)`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

	■ <code>'checkOtherWidth</code>: (Virtuoso Layout Suite EXL and higher tiers) The constraint applies when the width of the other shape is also less than the width parameter. Additionally, the width of the other shape is checked only in the direction orthogonal to the direction parameter, if specified. ■ <code>'otherWidthRanges</code>: The constraint applies only if the width of the other shape falls in one of these ranges. Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint
<code>'equalRectWidth</code>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the length of the end-of-line edge is equal to the width of the shape.
<code>'ignoreJoint</code>	(Virtuoso Layout Suite EXL and higher tiers) The constraint ignores violations for the joint inside the keepout region.
<code>'orthogonalPair</code>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to the shapes in orthogonal directions.

Examples

- [Example 1: endOfLineKeepout](#)
- [Example 2: endOfLineKeepout with widthRanges](#)
- [Example 3: endOfLineKeepout with cornerOnly and exceptFromFrontEdge](#)
- [Example 4: endOfLineKeepout with exceptSideWithin](#)
- [Example 5: endOfLineKeepout with forwardGap](#)
- [Example 6: endOfLineKeepout with orGroupID \(OR\)](#)
- [Example 7: endOfLineKeepout with orGroupID \(OR and AND\)](#)
- [Example 8: endOfLineKeepout with mask, twoSides, and diffMask](#)
- [Example 9: endOfLineKeepout with otherEndEol](#)
- [Example 10: endOfLineKeepout with exceptSameSideMetal](#)
- [Example 11: endOfLineKeepout with exceptNoOppSideMetal](#)
- [Example 12: endOfLineKeepout with exceptSpacing and exceptEolSpacing](#)
- [Example 13: endOfLineKeepout with eolSpacingRange and otherEolSpacingRange](#)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- [Example 14: endOfLineKeepout with jointAsEol](#)
- [Example 15: endOfLineKeepout with checkOtherWidth](#)
- [Example 16: endOfLineKeepout with equalRectWidth](#)
- [Example 17: endOfLineKeepout with ignoreJoint](#)

Virtuoso Technology Data Constraints Reference

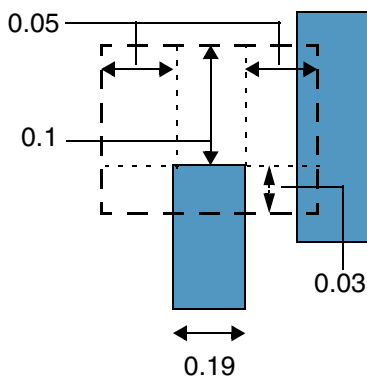
Spacing Constraints (One Layer)

Example 1: endOfLineKeepout

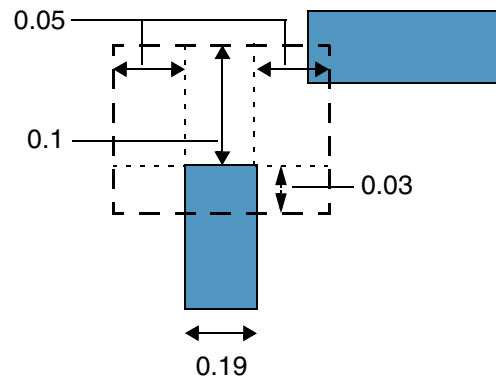
The keepout region for a wire with end-of-line width less than 0.2 and minimum EOL width less than or equal to 0.05 is defined by backward, side, and forward extension values of 0.03, 0.05, and 0.1, respectively. The constraint is violated if the corner or edge of a neighboring shape intersects the keepout region.

```
spacings(  
  ( endOfLineKeepout "Metal1" 'width 0.2 'minEolWidth 0.05  
    (0.03 0.05 0.1)  
  )  
) ;spacings
```

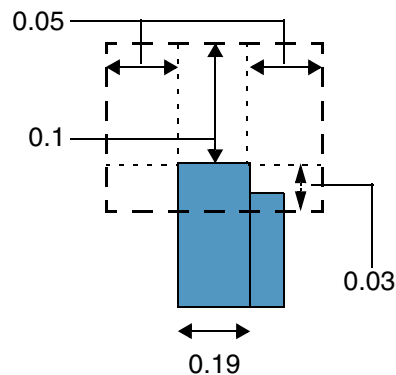
■ Metal1
□ keepout region



a) FAIL. The edge of a neighboring shape intersects the keepout region. If 'cornerOnly was defined, this would not be a violation.



b) FAIL. The corner of a neighboring shape intersects the keepout region.



c) FAIL. Any intersecting neighboring shape, including a same-metal shape, is a violation.

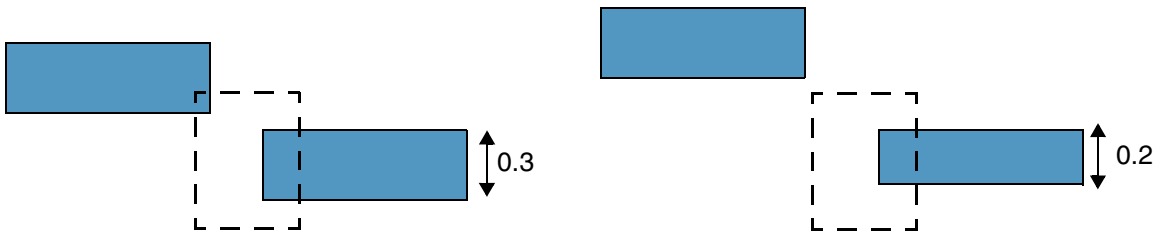
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 2: *endOfLineKeepout* with *widthRanges*

```
(endOfLineKeepout "Metal1"  
  'width 0.6  
  'widthRanges (">=0.3")  
  (0.4 0.8 0.5)  
)
```

■ Metal1



a) FAIL. The constraint applies because the end-of-line width of 0.3 is ≥ 0.3 and the neighbor wire overlaps the keepout window.

b) PASS. The constraint does not apply because the end-of-line width of 0.2 is not ≥ 0.3 .

Example 3: *endOfLineKeepout* with *cornerOnly* and *exceptFromFrontEdge*

The keepout region for a wire with end-of-line width less than 0.2 is defined by backward, side, and forward extension values of 0.03, 0.05, and 0.1, respectively. The constraint is violated if

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

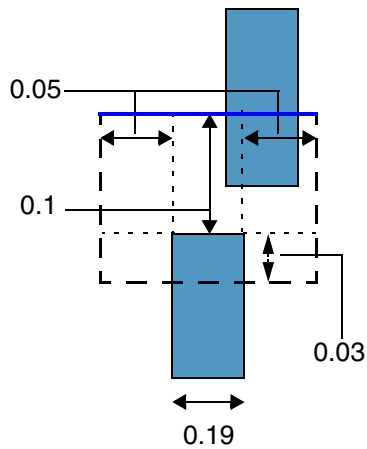
the corner of a neighboring shape, which is not coming from the front edge of the keepout region, intersects the keepout region.

```

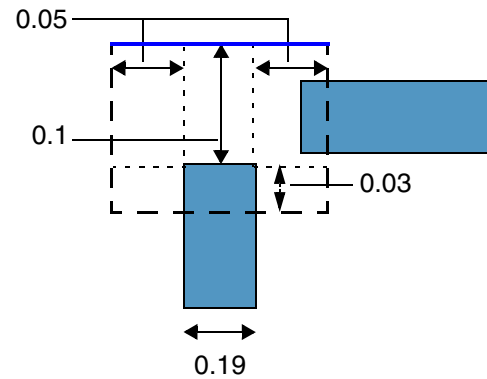
spacings (
  ( endOfLineKeepout "Metal1" 'width 0.2
    'cornerOnly
    'exceptFromFrontEdge
    (0.03 0.05 0.1)
  )
) ;spacings

```

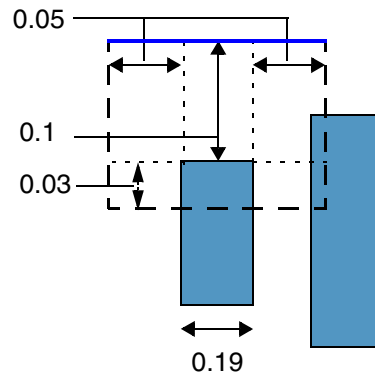
■ Metal1
 □ keepout region



a) PASS. The neighboring shape is coming from the front edge (the blue, solid line) of the keepout region.



b) FAIL. The corner of a neighboring shape, which is not coming from the front edge, intersects the keepout region.



c) FAIL. The corner of a neighboring shape, which is not coming from the front edge, intersects the keepout region.

Virtuoso Technology Data Constraints Reference

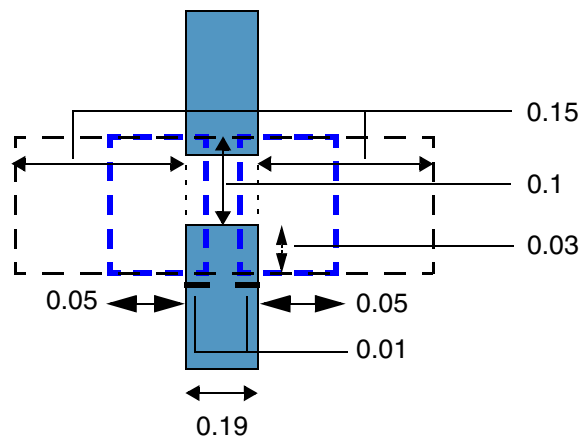
Spacing Constraints (One Layer)

Example 4: *endOfLineKeepout* with *exceptSideWithin*

The keepout region for a wire with end-of-line width less than 0.2 is defined by backward, side, and forward extension values of 0.03, 0.15, and 0.1, respectively. The constraint is violated if the corner of a neighboring shape, which is not coming from the front edge of the keepout region, intersects the keepout region, except when it overlaps the exempted regions.

```
spacings (  
  ( endOfLineKeepout "Metal1" 'width 0.2  
    'exceptSideWithin (-0.01 0.05)  
    (0.03 0.15 0.1)  
  )  
) ;spacings
```

- Metal1
- keepout region
- Exemption region



PASS. When 'exceptSideWithin is negative, a neighboring shape on the same track is exempted.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 5: endOfLineKeepout with forwardGap

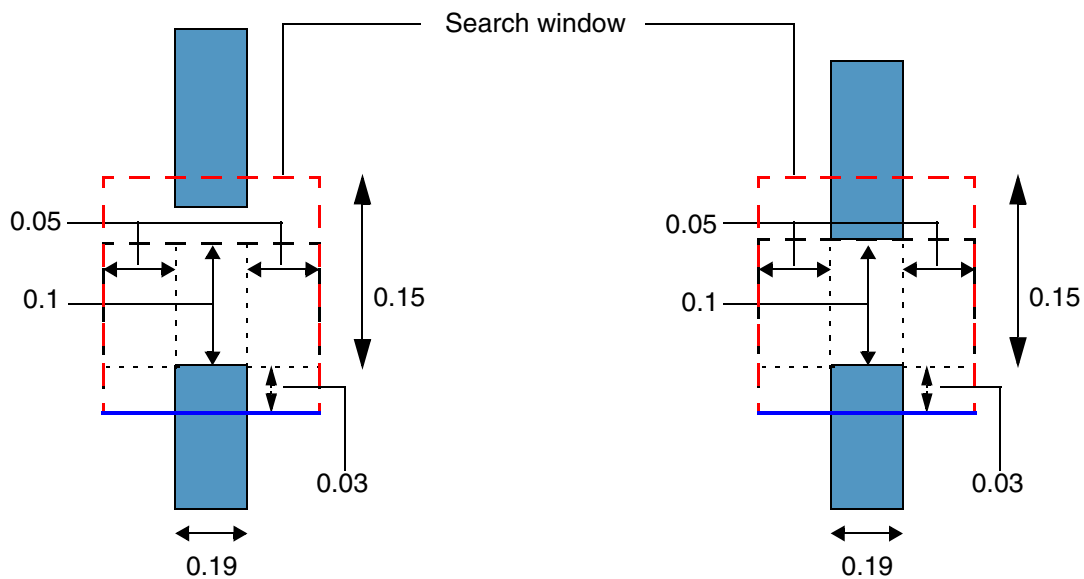
The keepout region for a wire with end-of-line width less than 0.2 is defined by backward, side, and forward extension values of 0.03, 0.05, and 0.1, respectively. If the distance between the wire and a face-to-face shape is less than 0.15, then the spacing between them must be 0.1 (the forward extension value).

```

spacings (
  ( endOfLineKeepout "Metal1" 'width 0.2
    'cornerOnly
    'exceptFromBackEdge 'forwardGap 0.15
    (0.03 0.05 0.1)
  )
) ;spacings

```

Metal1
 keepout region



a) FAIL. The spacing between the two face-to-face shapes is less than 0.15, and not equal to 0.1 (*forwardExt*). The blue, solid line represents the back edge of the keepout region.

b) PASS. The spacing between the two face-to-face shapes is equal to 0.1.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

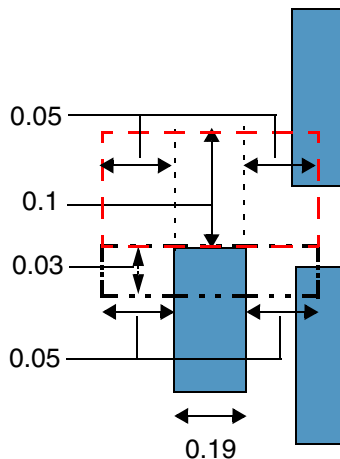
Example 6: endOfLineKeepout with orGroupID (OR)

Two keepout regions for a wire with end-of-line width less than 0.2 are defined by backward, side, and forward extension values of 0.03, 0.05, and 0.0, respectively, for region 1, and 0.0, 0.05, and 0.1, respectively, for region 2. The constraint is violated if all of the following conditions are met:

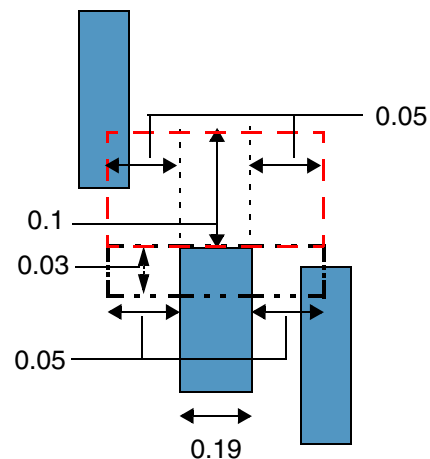
- The corner of a neighboring shape, which is not coming from the front edge of keepout region 1, intersects keepout region 1.
- The corner of a neighboring shape, which is not coming from the back edge of keepout region 2, intersects keepout region 2.
- Both neighboring shapes are on the same side of the end-of-line wire.

```
spacings (
  ( endOfLineKeepout "Metal1" 'width 0.2
    'cornerOnly
    'exceptFromFrontEdge
    'orGroupID 1
    (0.03 0.05 0.0)
  )
  ( endOfLineKeepout "Metal1" 'width 0.2
    'cornerOnly
    'exceptFromBackEdge
    'orGroupID 1
    (0.0 0.05 0.1)
  )
) ;spacings
```

■ Metal1
□ keepout region 1
□ keepout region 2



a) FAIL. Both constraints fail on the same side of the end-of-line wire.



b) PASS. Both constraints fail, but the failing neighboring shapes are on opposite sides of the end-of-line wire.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 7: endOfLineKeepout with orGroupID (OR and AND)

In the following example, a violation occurs if all of the first three rules fail or if both of the last two rules fail:

```
spacings (
  ( endOfLineKeepout "Metall" 'width 0.06
    'cornerOnly
    'exceptFromFrontEdge
    'orGroupID 1
    (0.00 0.08 0.12)
  )
  ( endOfLineKeepout "Metall" 'width 0.06
    'cornerOnly
    'exceptFromBackEdge
    'orGroupID 1
    (0.04 0.08 0.00)
  )
  ( endOfLineKeepout "Metall" 'width 0.06
    'cornerOnly
    'orGroupID 1
    (0.04 0.06 0.12)
  )
  ( endOfLineKeepout "Metall" 'width 0.09
    'cornerOnly
    'exceptFromFrontEdge
    'orGroupID 2
    (0.04 0.08 0.12)
  )
  ( endOfLineKeepout "Metall" 'width 0.09
    'cornerOnly
    'exceptFromBackEdge
    'orGroupID 2
    (0.04 0.00 0.12)
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 8: endOfLineKeepout with mask, twoSides, and diffMask

The keepout region for a wire with end-of-line width less than 0.2 is defined by backward, side, and forward extension values of 0.03, 0.05, and 0.1, respectively. The constraint applies only to a mask1 wire with two neighboring wires on a different mask, one on each side. The neighboring wires must also satisfy 'cornerOnly and 'exceptFromBackEdge.

```

spacings (
  ( endOfLineKeepout "Metal2" 'width 0.2
    'cornerOnly
    'exceptFromBackEdge
    'mask1 'twoSides 'diffMask
    (0.03 0.05 0.1)
  )
) ;spacings

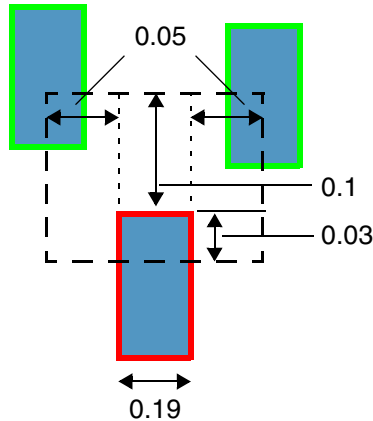
```

Metal1

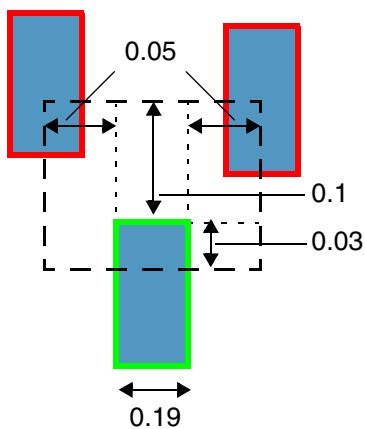
mask1

mask2

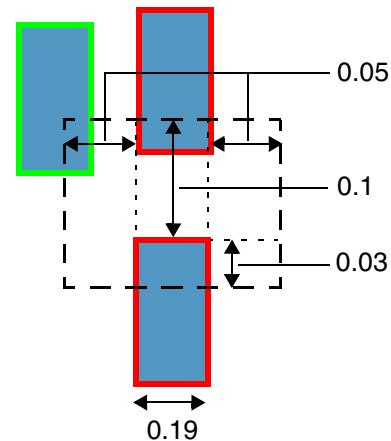
keepout region



a) FAIL. The constraint applies because the mask1 wire has two different-mask (mask2) wires that satisfy both 'cornerOnly and 'exceptFromBackEdge as neighbors, but is not met because the corners of these neighboring wires intersect the keepout region.



b) The constraint does not apply. The end-of-line wire must be a mask1 wire.



c) The constraint does not apply because only one neighboring wire inside the keepout region is on mask2.

Virtuoso Technology Data Constraints Reference

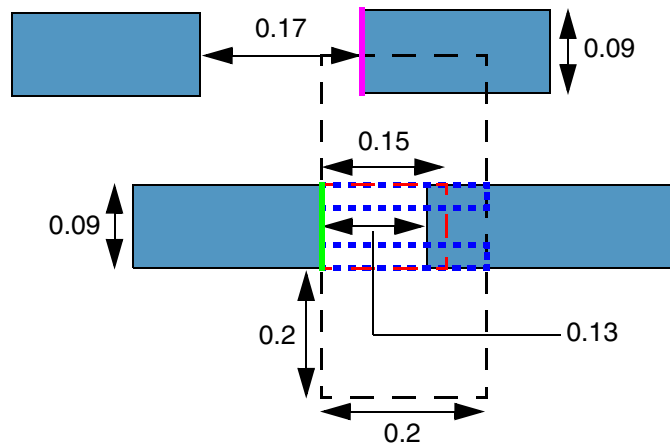
Spacing Constraints (One Layer)

Example 9: *endOfLineKeepout with otherEndEol*

Two `endOfLineKeepout` constraints in an OR group define two keepout regions for wires with end-of-line width less than 0.1; region 1 with backward, side, and forward extensions of 0.0, 0.2, and 0.2, respectively; and region 2 with backward, side, and forward extensions of 0.0, 0.0, and 0.15, respectively. A violation does not occur if an end-of-line edge that passes one of the constraints in the OR group is found inside the keepout region.

```
spacings (
  ( endOfLineKeepout "Metal1" 'width 0.1
    'cornerOnly
    'exceptFromBackEdge
    'exceptSideWithin (-0.01 0.0)
    'orGroupID 1 'otherEndEol
    (0.0 0.2 0.2)
  )
  ( endOfLineKeepout "Metal1" 'width 0.1
    'orGroupID 1
    (0.0 0.0 0.15)
  )
) ;spacings
```

- Metal1
- Keepout region 1
- Keepout region 2
- Exemption region



PASS. The pink edge is an end-of-line edge found inside the keepout region (region 1) of the green end-of-line edge. When the green edge is the trigger edge, both `endOfLineKeepout` constraints in the OR group fail—a corner of the top wire on the right intersects the front edge of keepout region 1 (in black), and, therefore, fails the first `endOfLineKeepout` constraint; the bottom wire on the right intersects keepout region 2 (in red), and, therefore, fails the second `endOfLineKeepout` constraint. Because the "other" end-of-line edge (in pink) inside the keepout region passes the second constraint in the OR group, the green end-of-line edge passes the constraint, even though it fails both constraints in the OR group.

Virtuoso Technology Data Constraints Reference

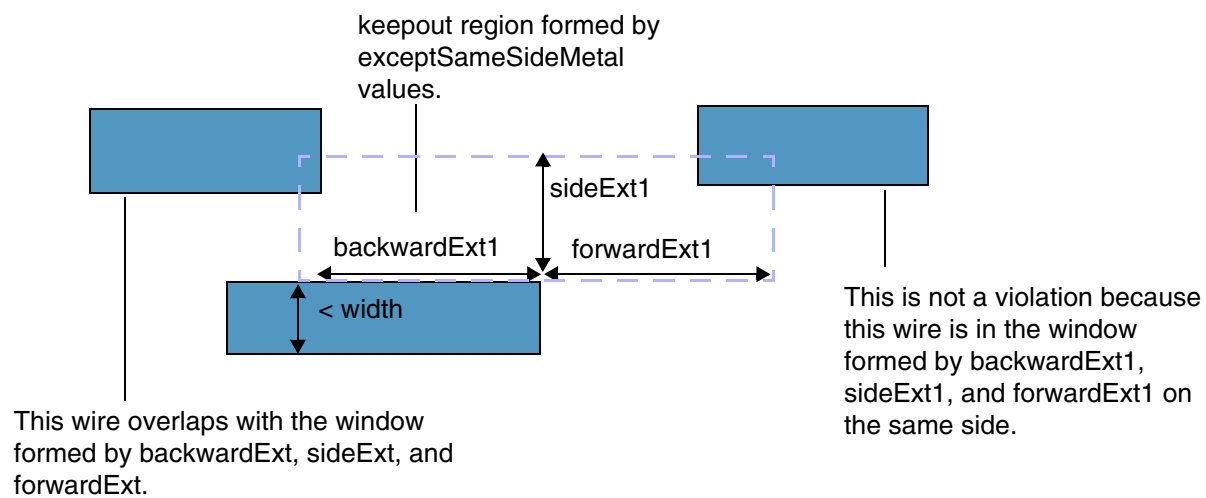
Spacing Constraints (One Layer)

Example 10: endOfLineKeepout with exceptSameSideMetal

The following diagram illustrates the exceptSameSideMetal parameter.

```
spacings (
  ( endOfLineKeepout "Metal1"
    'width w
    'exceptSameSideMetal (backwardExt1 sideExt1 forwardExt1)
    (backwardExt sideExt forwardExt)
  )
) ;spacings
```

■ Metal1

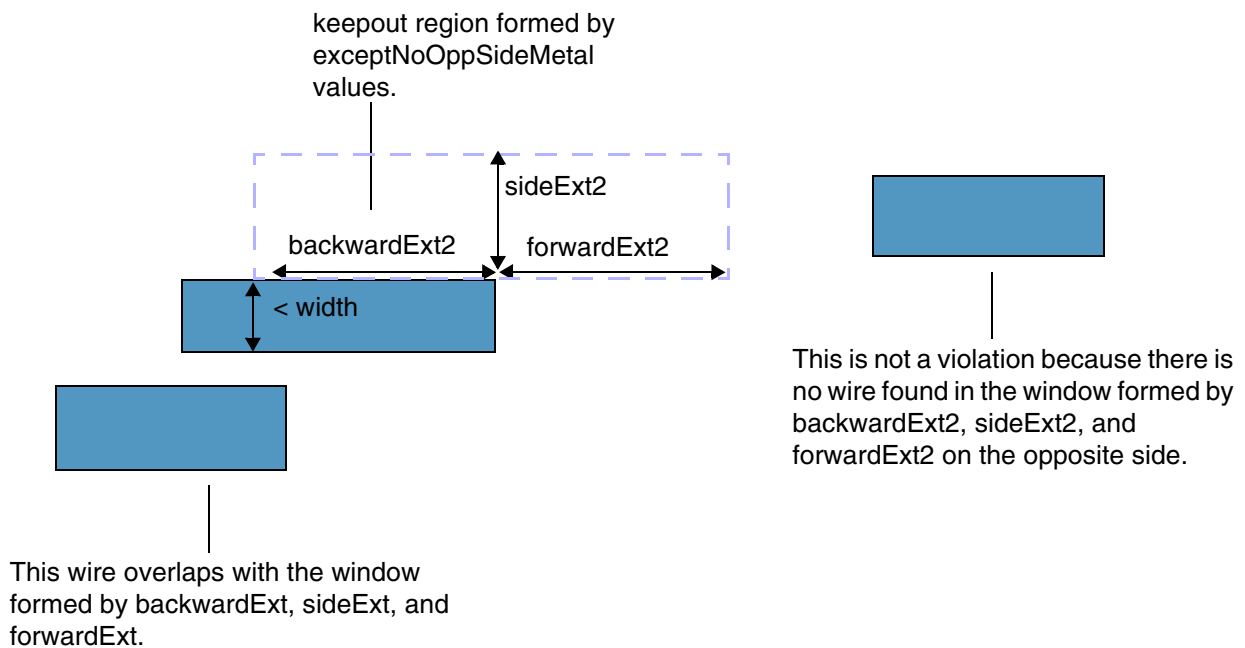


Example 11: endOfLineKeepout with exceptNoOppSideMetal

The following diagram illustrates the exceptNoOppSideMetal parameter.

```
spacings (
    ( endOfLineKeepout "Metal1"
      'width w
      'exceptNoOppSideMetal (backwardExt2 sideExt2 forwardExt2)
      (backwardExt sideExt forwardExt)
    )
) ;spacings
```

■ Metal1



Example 12: endOfLineKeepout with exceptSpacing and exceptEolSpacing

The first constraint applies only when the neighbor wire has another wire with an end-of-line spacing of exactly 0.06. The second constraint applies only when there is a cut on either Via1 or Via2 within 0.05 from the end-of-line edge.

```
spacings (
    (endOfLineKeepout "Metal1" 'width 0.08
      'cornerOnly 'exceptSpacing 0.06
      (0.03 0.06 0.07))
    (endOfLineKeepout "Metal1" 'width 0.08
      'cornerOnly 'exceptEolSpacing 0.06
      'withinCut 0.05 'cutLayers ("Via1" "Via2")
      (0.03 0.06 0.07))
) ;spacings
```


Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

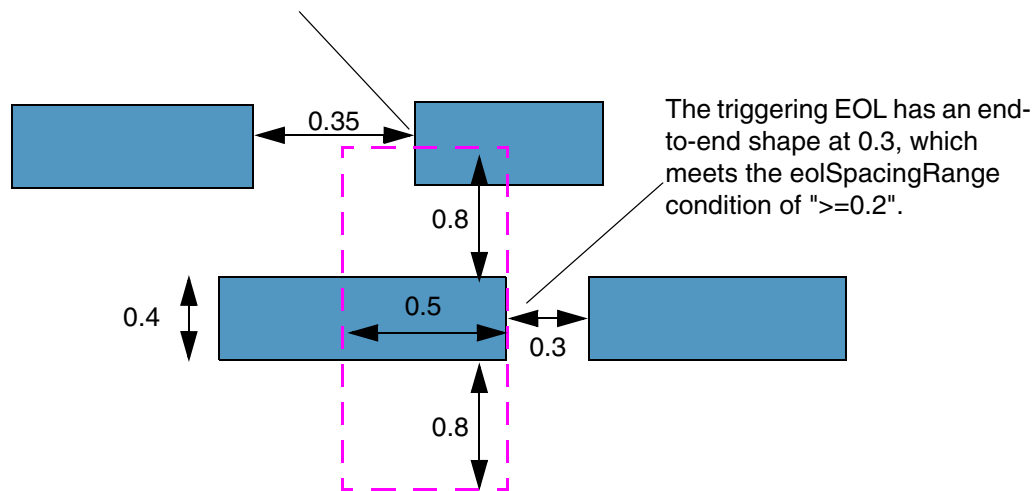
Example 13: *endOfLineKeepout* with *eolSpacingRange* and *otherEolSpacingRange*

The following examples illustrate the *eolSpacingRange* and *otherEolSpacingRange* parameters.

```
spacings (  
  ( endOfLineKeepout "Metal1"  
    'width 0.6  
    'horizontal  
    'cornerOnly  
    'exceptFromBackEdge  
    'eolSpacingRange ">=0.2"  
    'otherEolSpacingRange ">=0.3"  
    (0.5 0.8 0)  
  )  
) ;spacings
```

■ Metal1
□ EOL keepout region 1

The EOL edge intersects with the keepout region and has an end-to-end shape at 0.35 spacing, which meets the *otherEolSpacingRange* condition of " ≥ 0.3 ".



FAIL. Both the triggering EOL and the neighboring EOL edge have an end-to-end shape meeting the *eolSpacingRange* and *otherEolSpacingRange* values but the corner of a neighboring shape is inside the EOL keepout box.

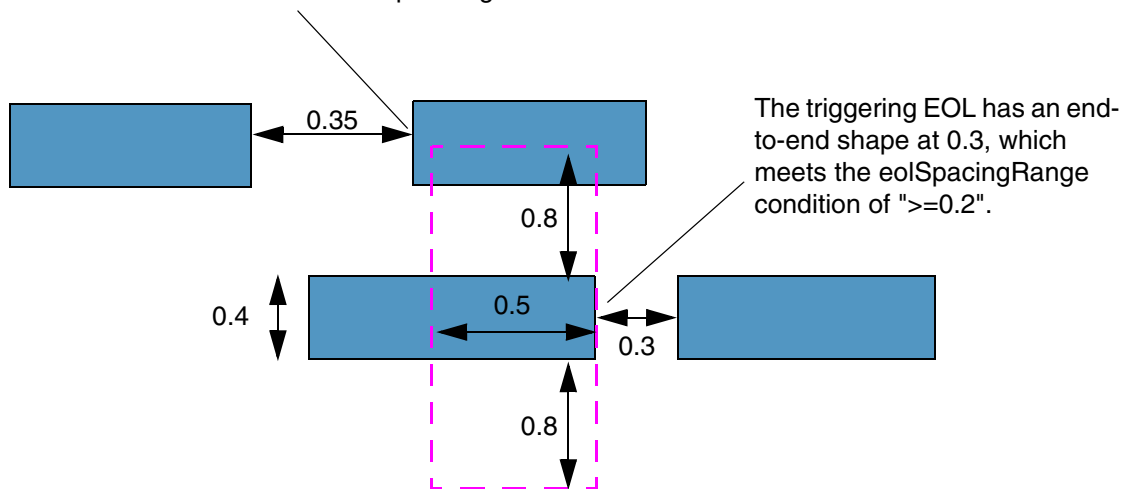
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
spacings (
  ( endOfLineKeepout "Metal1"
    'width 0.6
    'horizontal
    'eolSpacingRange ">=0.2"
    'otherEolSpacingRange ">=0.3"
    (0.5 0.8 0)
  )
) ;spacings
```

■ Metal1
□ EOL keepout region 1

The EOL edge of the neighboring shape does not intersect or lie within the keepout region.



PASS. The constraint does not apply because the EOL edge of the neighboring shape does not lie within the keepout region.

Virtuoso Technology Data Constraints Reference

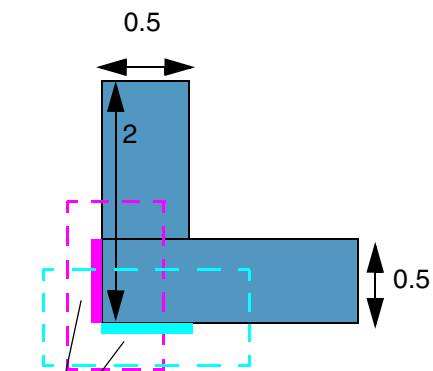
Spacing Constraints (One Layer)

Example 14: *endOfLineKeepout* with *jointAsEol*

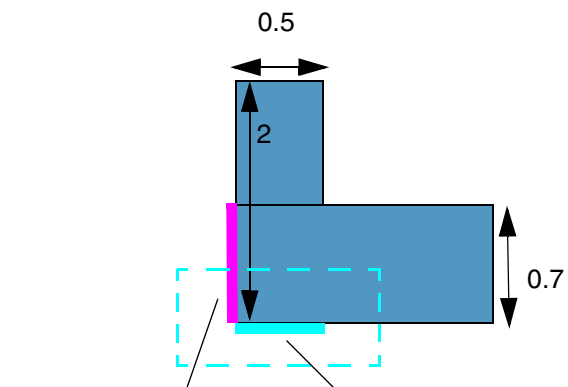
Two edges at a joint of two intersecting shapes are treated as end-of-line when one of the edges of the intersecting shapes that forms the joint edge has width less than 0.6.

```
spacings (
  ( endOfLineKeepout "Metal1"
    'width 0.6
    'jointAsEol
    (0.4 0.8 0.5)
  )
) ;spacings
```

■ Metal1
□ keepout region 1
□ keepout region 2

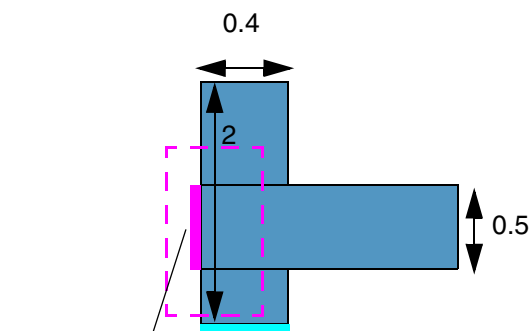


Consider as EOL because 0.5 is less than 0.6.



Do not consider as EOL because 0.7 is greater than 0.6.

Consider as EOL because 0.5 is less than 0.6.



Consider as EOL because 0.5 is less than 0.6.

Virtuoso Technology Data Constraints Reference

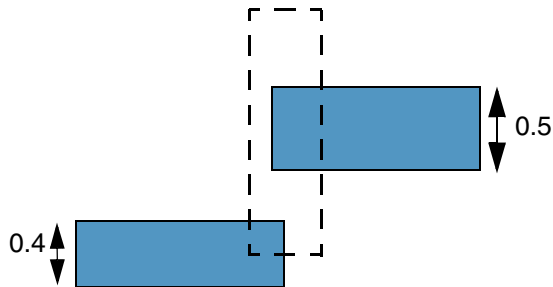
Spacing Constraints (One Layer)

Example 15: *endOfLineKeepout* with *checkOtherWidth*

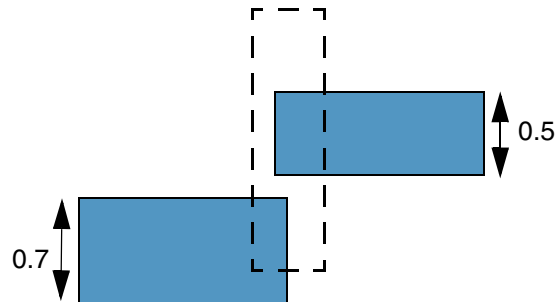
The constraint applies when the width of the other shape is also less than 0.6. Additionally, the width of the other shape is checked only in the direction orthogonal to the direction parameter, which is specified here as horizontal.

```
spacings (
  ( endOfLineKeepout "Metal1"
    'width 0.6
    'horizontal
    'checkOtherWidth
    (0.4 0.8 0.5)
  )
)
```

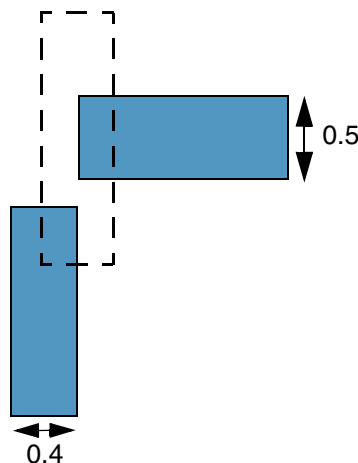
■ Metal1
□ Keepout region



FAIL. The constraint applies because the width of the other shape to the left is 0.4, which is less than 0.6, and meets the width requirement. The constraint fails because the width is in the vertical direction, which is orthogonal to the specified direction.



PASS. The constraint does not apply because the width of the other shape to the left is 0.7, which is greater than 0.6, and does not meet the width requirement.



PASS. The constraint does not apply because the width of the other shape below, though less than 0.6, is in same direction as the specified direction parameter.

Virtuoso Technology Data Constraints Reference

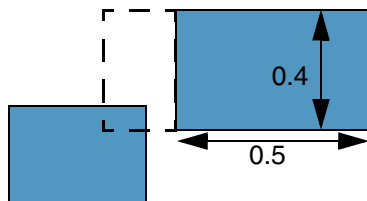
Spacing Constraints (One Layer)

Example 16: endOfLineKeepout with equalRectWidth

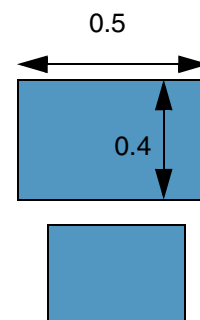
The constraint applies only if the length of the end-of-line edge is equal to 0.6.

```
spacings(  
  ( endOfLineKeepout "Metal1"  
    'width 0.6  
    'equalRectWidth  
    (0.0 0.0 0.5)  
  )  
) ;spacings
```

■ Metal1
□ Keepout region



FAIL. The constraint applies to the end-of-line edge because the end-of-line width is 0.4, which is equal to the shape width and fails because the other shape intersects the keepout region.



PASS. The constraint does not apply to the end-of-line edge because the end-of-line width is 0.5, which is greater than the shape width of 0.4.

Virtuoso Technology Data Constraints Reference

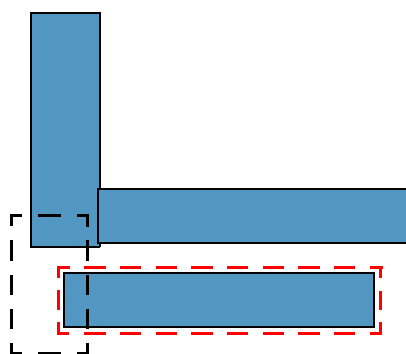
Spacing Constraints (One Layer)

Example 17: endOfLineKeepout with ignoreJoint

The constraint does not apply when the end-of-line keepout region contains only wire joints.

```
spacings (  
  ( endOfLineKeepout "M0"  
    'width 0.6  
    'ignoreJoint  
    (0.4 0.8 0.5)  
  )  
) ;spacings
```

■ M0



PASS. Joint is ignored.

eolViaKeepout

```
orderedSpacings(  
  ( eolViaKeepout tx_metal tx_cut  
    {'mask1 | 'mask2 | 'mask3}  
    'minEolWidth f_minEolWidth  
    'otherWidth f_otherWidth  
    'otherExtension f_otherExtension  
    ['exceptBackSideAligned]  
    (f_backwardExt f_sideExt f_forwardExt)  
  )  
); orderedSpacings
```

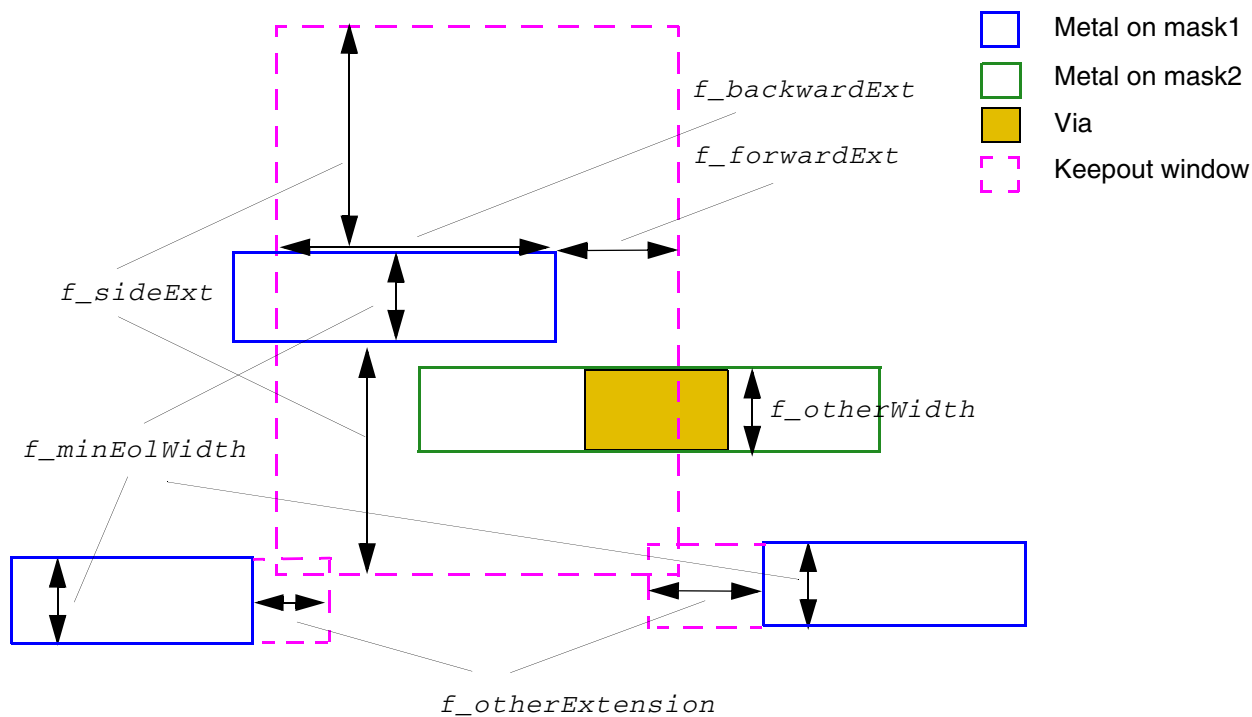
(Virtuoso Layout Suite EXL and higher tiers) Specifies a keepout region against a cut if the following conditions are met:

- The cut does not touch or overlap with the region if it is in a wire with width less than or equal to the specified value and with mask different from the mask of the end-of-line.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

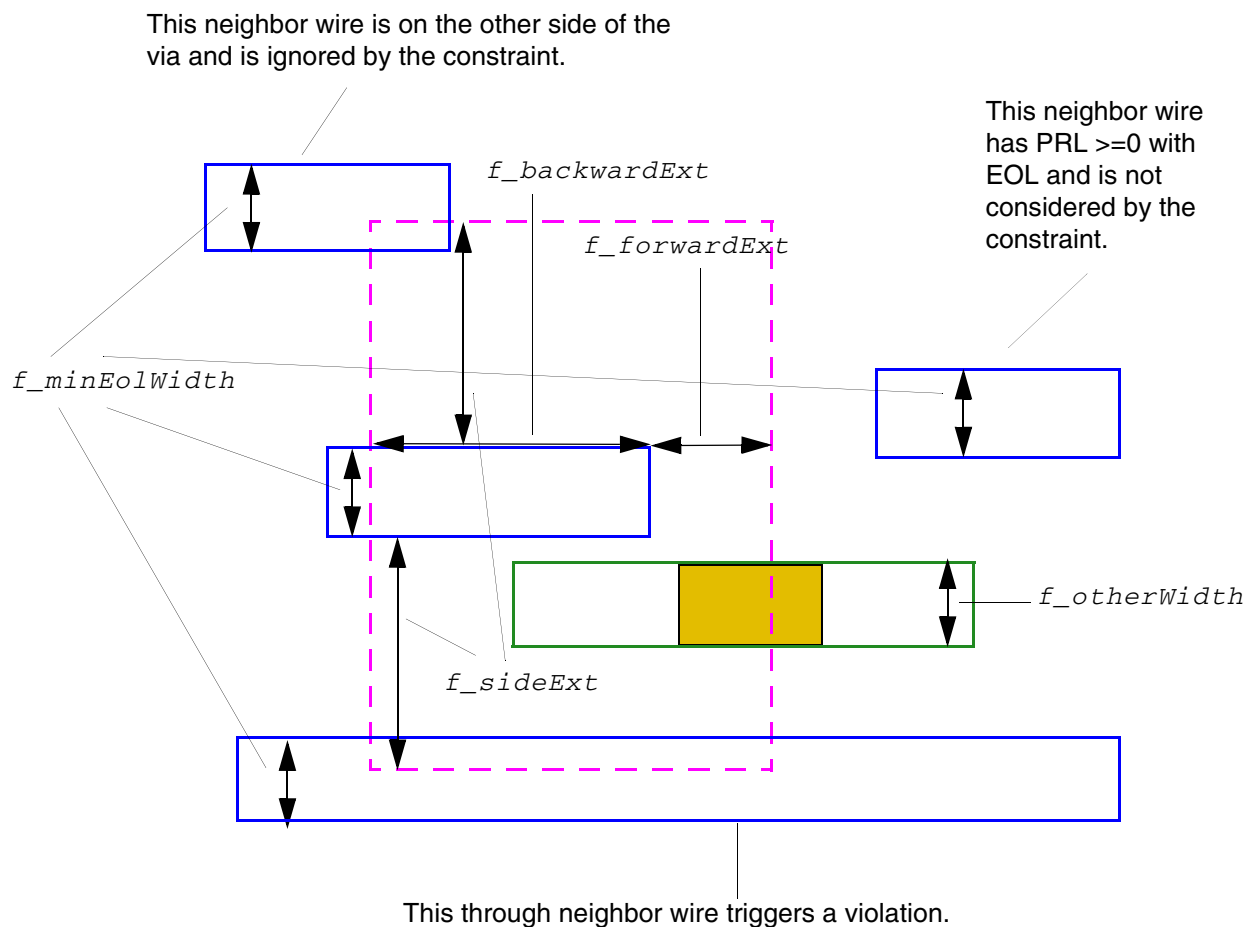
- A same-mask neighbor wire has an extension overlapping with the region.



If either bottom neighbor wire fulfills the $f\_otherExtension$ condition, it would be a violation for a wire of another mask containing an above or below cut touching or overlapping with the keepout window.

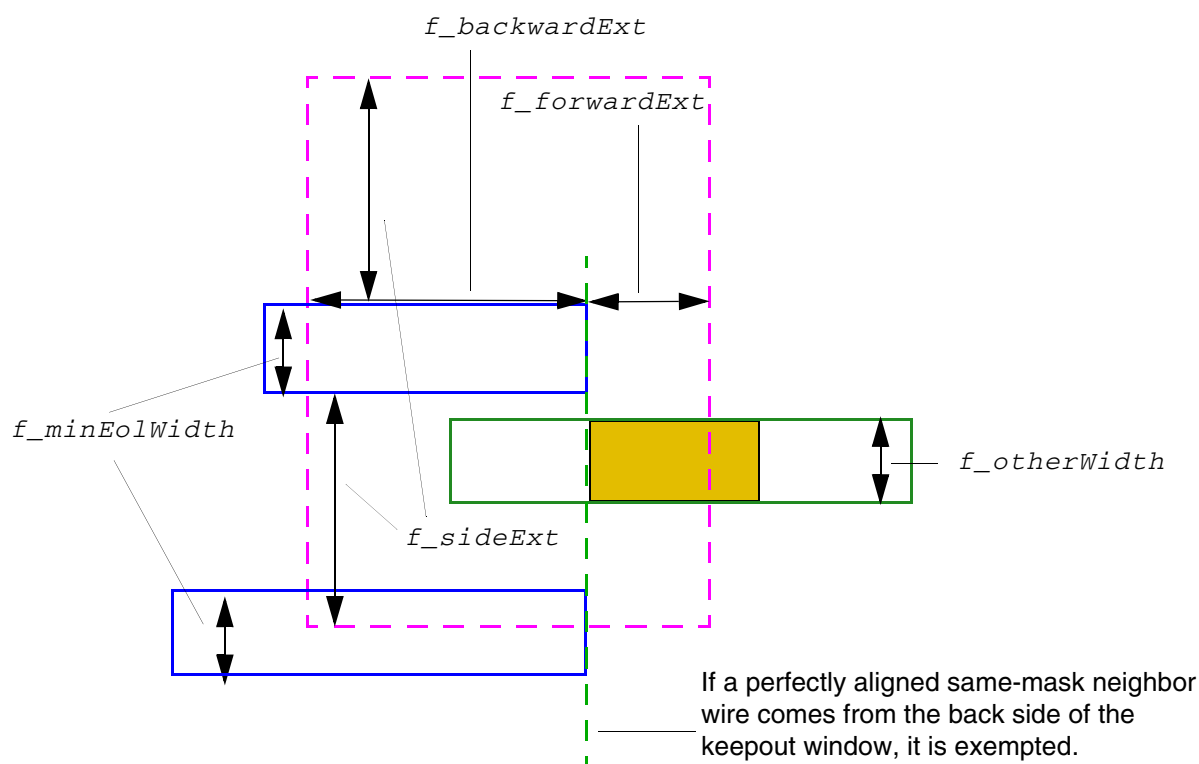
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



Values

tx_metal

The metal layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

tx_cut

The cut layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

f_backwardExt f_sideExt f_forwardExt

The amount of extension from the corners of the end-of-line edge:

- *f_backwardExt* indicates extension in the backward direction.
- *f_sideExt* indicates extension to the side.
- *f_forwardExt* indicates extension in the forward direction.

Parameters

'minEolWidth *f_minEolWidth*

The constraint applies when the end-of-line has width less than this value.

'mask1 | 'mask2 | 'mask3

The constraint applies when the end-of-line shape is on the specified mask.

'otherWidth *f_otherWidth*

The constraint applies to different-mask wires that have width less than or equal to this value and a cut that overlaps or touches the keepout region.

'otherExtension *f_otherExtension*

The constraint applies when there is a same-mask neighbor wire with the specified extension value at the end-of-line overlapping the keepout region.

'exceptBackSideAligned

The constraint does not apply if the neighbor wire is aligned to the end-of-line from the back.

Virtuoso Technology Data Constraints Reference

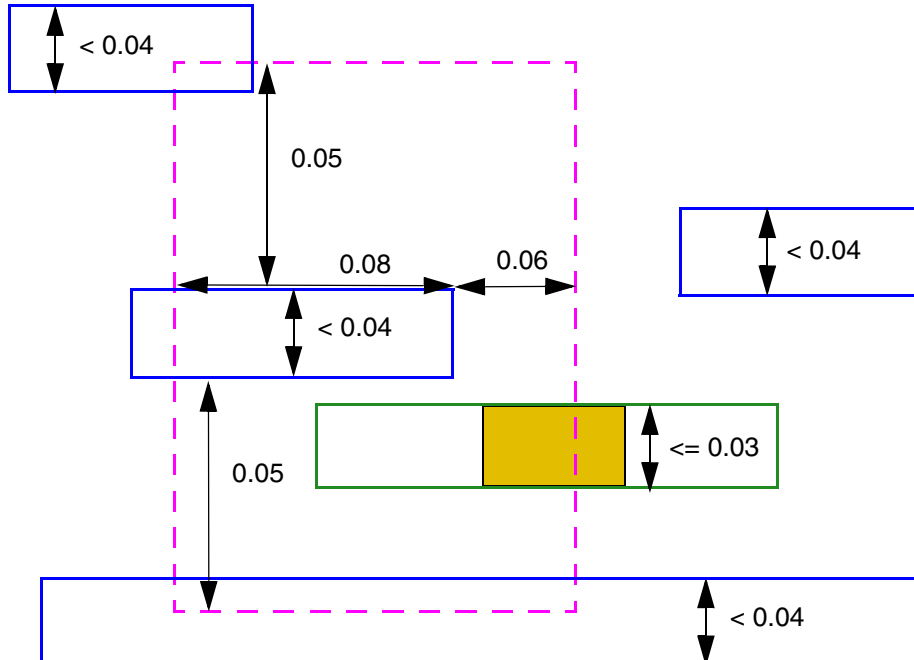
Spacing Constraints (One Layer)

Example

Specifies a keepout region between Metal1 and Via1 if the specified conditions are met, except when the neighbor wire is aligned to the end-of-line from the back.

```
orderedSpacings(  
  (eolViaKeepout "Metal1" "Via1"  
    'mask1  
    'minEolWidth 0.04  
    'otherExtension 0.01  
    'otherWidth 0.03  
    'exceptBackSideAligned  
    (0.08 0.05 0.06))  
)  
;orderedSpacings
```

-  Metal on mask1
-  Metal on mask2
-  Via1
-  Keepout window



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

forbiddenEdgePitchRange

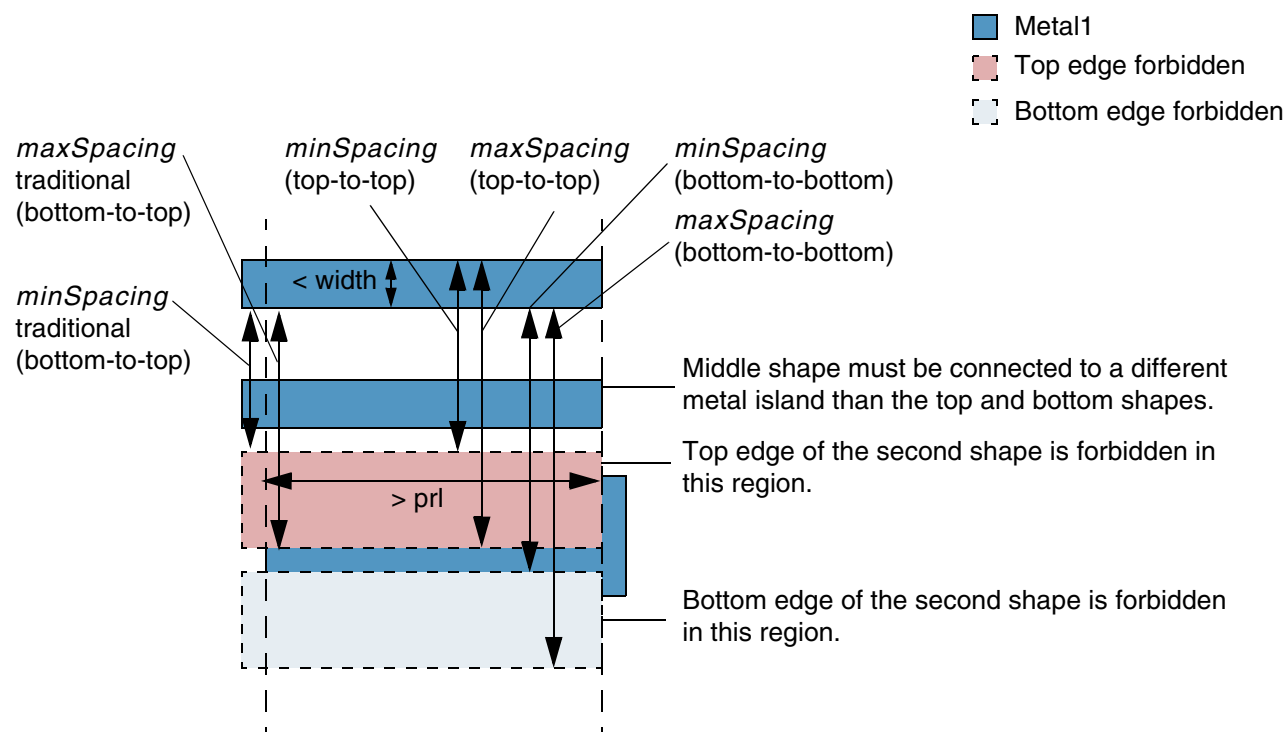
```
spacings (
  ( forbiddenEdgePitchRange tx_layer
    'width f_width ['prl f_prl]
    ['minWidth f_minWidth
      ['outerWidth f_outerWidth]
      ['outerWithin f_outerWithin]
      ['outerSameMask]
    ]
    ['parallelWithin f_parallelWithin
      | 'exactSpacingNeighbor f_exactSpacing
        ['spanLength f_spanLength ['within f_within]]
      | 'neighborSpacingRanges g_neighborSpacingRanges
    ['exceptWireWidthSpacingList (g_exactWidth g_spacing)...]
    ['ignoreMiddleWire (g_Width g_spacing)]
    ]
    ['secondForbiddenRange g_secondForbiddenRange]
    ['sameMask | 'mask1 | 'mask2 | 'mask3]
    ['exactWireWidth]
    ['otherWidth f_otherWidth]
    (g_range)
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Restricts the spacing between the left-to-left and right-to-right edges (or between top-to-top and bottom-to-bottom edges) of two shapes on a layer if a shape from a different metal island lies between the two shapes. The constraint applies only if the width of the first shape is less than the specified value.

Optionally, you can specify that the common projected parallel run length between the two edges and the in-between shape must be greater than a particular value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

g_range

The forbidden spacing range, that is, the spacing between shapes must not fall in this range.

The forbidden spacing range is specified using the format:

$[f\_min1 \ f\_max1]$

where, $f\_min1 \leq \text{spacing} \leq f\_max1$

Type: Floating-point values specifying a range of spacings that are not allowed

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

<code>'width <i>f_width</i></code>	The constraint applies only if the width of the first shape is less than this value.
<code>'minWidth <i>f_minWidth</i></code>	(Virtuoso Advanced Node for Layout Only) The constraint applies to a shape with width less than <i>width</i> and greater than <i>minWidth</i> if the shape is sandwiched by two shapes. The forbidden spacing applies between the middle shape and the outer shapes.
<code>'outerWidth <i>f_outerWidth</i></code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the width of the two outer shapes is greater than this value.
<code>'outerWithin <i>f_outerWithin</i></code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the distance between the two outer shapes is less than this value.
<code>'outerSameMask</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the two outer shapes are on the same mask. The mask applied to the middle shape is ignored. Type: Boolean
<code>'prl <i>f_prl</i></code>	The constraint applies only if the parallel run length between the two edges and the in-between shape is greater than this value.
<code>'parallelWithin <i>f_parallelWithin</i></code>	The constraint applies only if the first shape has neighboring shapes on both sides at a distance less than this value. Note: The spacing between the two shapes is measured from the bottom/top edge to the top/bottom edge (or from the right/left edge to the left/right edge).
<code>'exactSpacingNeighbor <i>f_exactSpacing</i></code>	

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The constraint applies only if the first shape has a neighboring shape at a distance equal to this value on the side opposite to the one on which spacing applies.

Note: The spacing between the two shapes is measured from the bottom/top edge to the top/bottom edge (or from the right/left edge to the left/right edge). A shape from a different metal island need not be present between the two shapes.

`'spanLength f_spanLength`

The constraint applies only if the span length of the second wire is less than this value.

Note: Span length is measured perpendicular to the parallel run length.

`'within f_within`

The constraint applies only if the second shape has a neighboring shape at a distance less than this value from it on either side.

`'neighborSpacingRanges g_neighborSpacingRanges`

The constraint applies only if the spacing between the first shape and a neighboring shape falls in this range.

Type: Floating-point values specifying a range of spacings that trigger the constraint

`'exceptWireWidthSpacingList (g_exactWidth g_spacing)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint specifies that neighbour wires, with at least one of them not in the forbidden spacing range, having width exactly equal to the given value with PRL greater than 0 and the spacing less than the given value are merged as if the gap between the wires is filled. For this parameter, `g_exactWidth` is the width value and `g_spacing` is the spacing value.

`'ignoreMiddleWire (g_width g_spacing)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint specifies that if there is a wire with a width not the same as the value of `g_width` or a spacing to the triggering wire not the same as the value of `g_spacing`, between the triggering wire and the forbidden spacing range wire, that wire is ignored.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'secondForbiddenRange *g_secondForbiddenRange*

A second forbidden spacing range.

The forbidden spacing range is specified using the format:

$[f\_min2 \ f\_max2]$

where, $f\_min2 \leq \text{spacing} \leq f\_max2$

Note: This second forbidden spacing range can be specified only if 'parallelWithin or 'exactSpacingNeighbor is specified.

Type: Floating-point values specifying an additional range of spacings that are not allowed

'sameMask

The constraint applies only to shapes on the same mask.

Note: The shapes between which forbidden spacing is checked must be on the same mask; the mask applied to neighboring shapes is ignored.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'mask1 | 'mask2 | 'mask3

(Virtuoso Advanced Node for Layout Only) The constraint applies only to the shapes on the specified mask.

Type: Boolean

'exactWireWidth

The constraint applies only if the width of the first shape is equal to *width*.

Type: Boolean

'otherWidth *f_otherWidth*

The constraint applies only if the width of the second shape is greater than or equal to this value.

Examples

- Example 1: forbiddenEdgePitchRange with width and prl
- Example 2: forbiddenEdgePitchRange with width, prl, and neighborSpacingRanges
- Example 3: forbiddenEdgePitchRange with width, prl, parallelWithin, and secondForbiddenRange
- Example 4: forbiddenEdgePitchRange with width, prl, exactSpacingNeighbor, and spanLength
- Example 5: forbiddenEdgePitchRange with width, exactSpacingNeighbor, exactWireWidth, and otherWidth
- Example 6: forbiddenEdgePitchRange with width, prl, exactSpacingNeighbor, spanLength, and within
- Example 7: forbiddenEdgePitchRange with width, minWidth, outerWidth, outerWithin, sameMask
- Example 8: forbiddenEdgePitchRange with width, minWidth, outerWidth, outerWithin, outerSameMask

Virtuoso Technology Data Constraints Reference

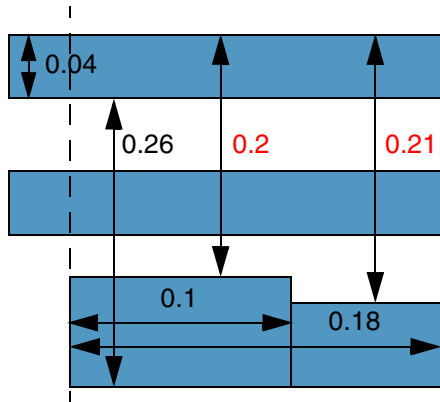
Spacing Constraints (One Layer)

Example 1: *forbiddenEdgePitchRange* with width and prl

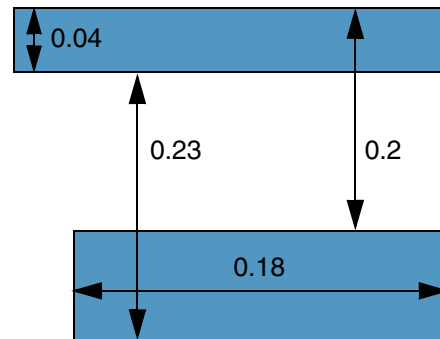
The top-to-top and bottom-to-bottom edge spacing between a Metal1 wire with width less than 0.05 and another Metal1 wire with which it has a parallel run length greater than 0.15 must not be greater than or equal to 0.2 and less than or equal to 0.25 if a wire from a different metal island lies between them.

```
spacings (  
  ( forbiddenEdgePitchRange "Metal1"  
    'width 0.05 'prl 0.15  
    "[0.2 0.25]"  
  )  
) ;spacings
```

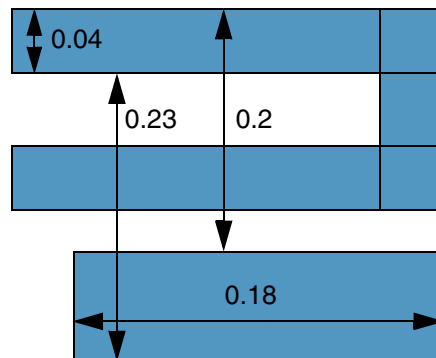
■ Metal1



a) FAIL. The width of the topmost wire is 0.04 (<0.05) and the cumulative parallel run length between the three wires is 0.18 (>0.15). However, the top-to-top spacings of 0.2 and 0.21 are in the forbidden spacing range. If the top-to-top spacing to one of the bottom wires was less than 0.2 or greater than 0.25, the constraint would pass.



b) The constraint does not apply because there is no wire between the top and bottom wires.



c) The constraint does not apply because the top and the middle wires are connected (that is, they are part of the same metal island).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

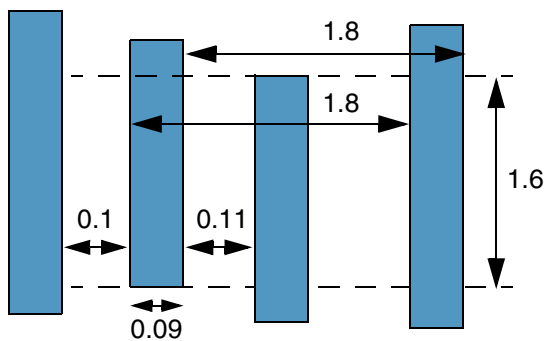
Example 2: forbiddenEdgePitchRange with width, prl, and neighborSpacingRanges

The left-to-left and right-to-right edge spacing between two Metal1 wires must not be greater than or equal to 2.0 and less than or equal to 3.0 if the following conditions are met:

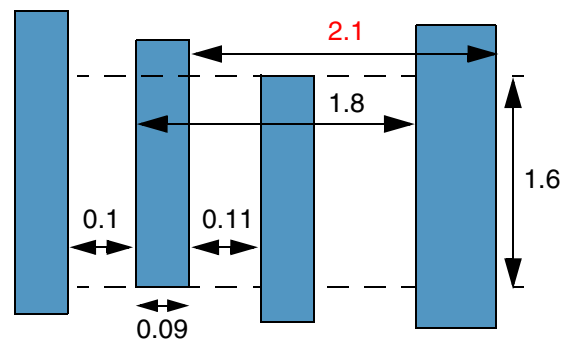
- The Metal1 wire under consideration has width less than 0.1.
- The parallel run length between the two Metal1 wires is greater than 1.5
- A wire from a different metal island with width equal to 0.1 or 0.15 or greater than or equal to 0.2 and less than or equal to 0.3 lies between the two Metal1 wires.

```
spacings (
    ( forbiddenEdgePitchRange "Metal1"
      'width 0.1 'prl 1.5
      'neighborSpacingRanges (0.1 0.15 "[0.2 0.3]")
      "[2.0 3.0]"
    )
) ;spacings
```

■ Metal1



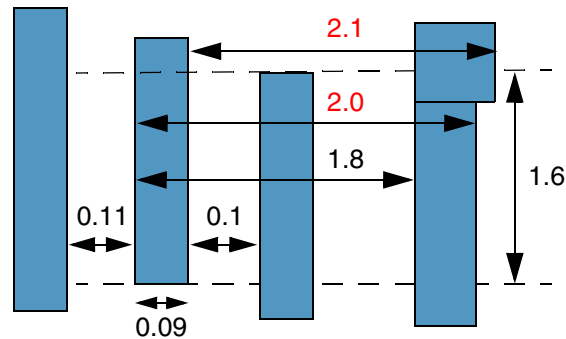
a) PASS. The target wire (second from left) is 0.09 (<0.1) wide and has a neighboring wire 0.1 away from it (the one on the left). The parallel run length between the wires is 1.6 (>1.5). The left-edge-to-left-edge and right-edge-to-right-edge spacing is 1.8, which lies outside the forbidden range.



b) FAIL. The left-edge-to-left-edge spacing is 1.8 (outside the forbidden range), but the right-edge-to-right-edge spacing of 2.1 lies in the forbidden range.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) FAIL. The left-edge-to-left-edge spacing is 1.8 (outside the forbidden range), but the right-edge-to-right-edge spacings of 2.0 and 2.1 lie in the forbidden range. If the right-edge-to-right-edge spacing to one of the rightmost wires was less than 2.0 or greater than 3.0, the constraint would pass.

Example 3: forbiddenEdgePitchRange with width, prl, parallelWithin, and secondForbiddenRange

The top-to-top and bottom-to-bottom edge spacing between two Metal1 wires that have a wire from a different metal island between them must not be greater than or equal to 0.2 and less than or equal to 0.25 if the following conditions are met:

- The Metal1 wire under consideration has width less than 0.05 and has neighboring wires on both sides at a distance less than 0.06 from it.
- The parallel run length between the two Metal1 wires is greater than 0.15.

Top-to-top and bottom-to-bottom edge spacing greater than or equal to 0.3 and less than or equal to 0.33 is also not allowed.

Virtuoso Technology Data Constraints Reference

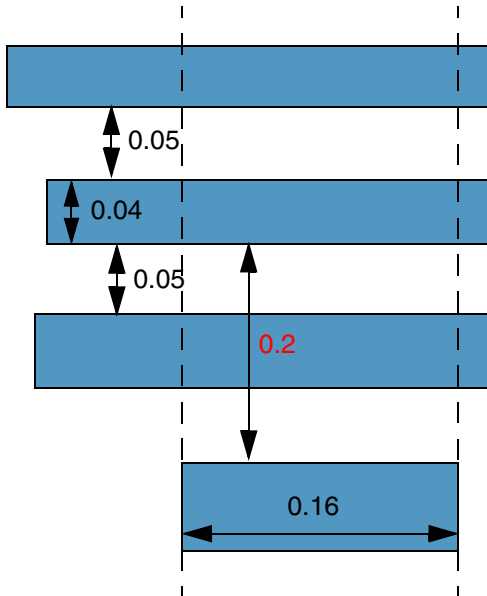
Spacing Constraints (One Layer)

```

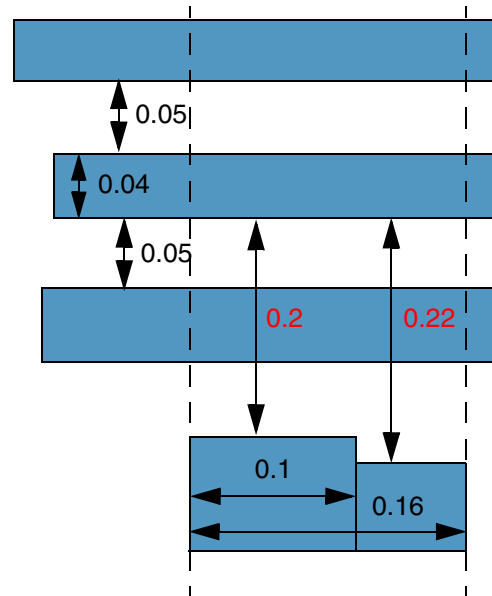
spacings (
  ( forbiddenEdgePitchRange "Metal1"
    'width 0.05 'prl 0.15
    'parallelWithin 0.06
    'secondForbiddenRange "[0.3 0.33]"
    "[0.2 0.25]"
  )
) ;spacings

```

■ Metal1



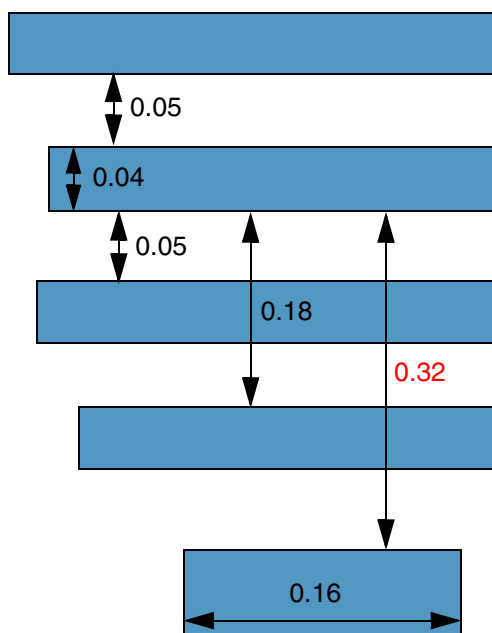
a) FAIL. The target wire (second from top) is 0.04 (<0.05) wide and has two neighboring wires at a distance of 0.05 (<0.06) from it. The parallel run length between the wires is 0.16 (>0.15). However, the spacing between the target wire and the bottommost wire is 0.2, which lies in the forbidden range.



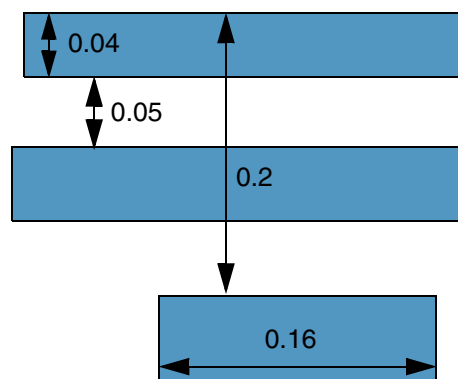
b) FAIL. The constraint applies. However, the spacing between the target wire (second from top) and the bottommost wire is 0.2 and 0.22, which lie in the forbidden range.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) FAIL. The spacing between the target wire (second from top) and the bottommost wire is 0.32, which lies in the second forbidden range.



d) The constraint does not apply because the target wire has only one neighboring wire at a distance 0.05 (< 0.06) from it.

Virtuoso Technology Data Constraints Reference

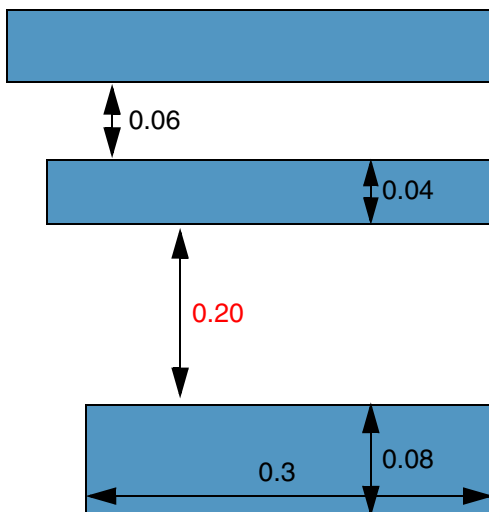
Spacing Constraints (One Layer)

Example 4: forbiddenEdgePitchRange with width, prl, exactSpacingNeighbor, and spanLength

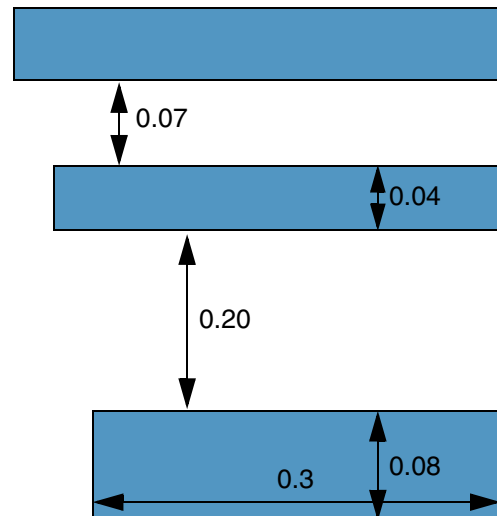
The spacing between two Metal1 wires must not be greater than or equal to 0.20 and less than or equal to 0.25 if the following conditions are met:

- The first wire is less than 0.05 wide and has a neighboring wire at a distance equal to 0.06 from it.
- The second wire has span length less than 0.09.
- The common projected parallel run length between the two wires is greater than 0.04.

```
spacings (                                     ■ Metal1
  ( forbiddenEdgePitchRange "Metal1"
    'width 0.05 'prl 0.04
    'exactSpacingNeighbor 0.06 'spanLength 0.09
    "[0.20 0.25]"
  )
) ;spacings
```



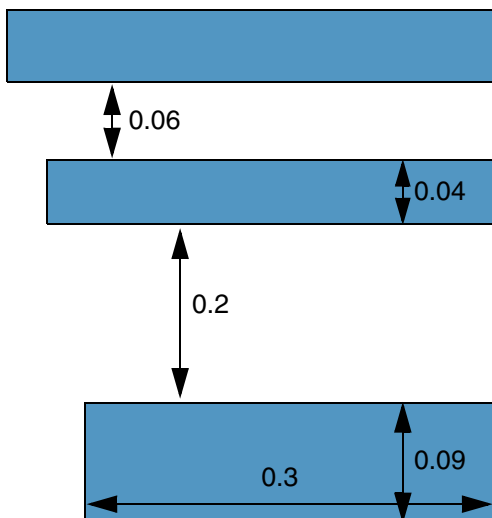
a) FAIL. The wire with width 0.04 (<0.05) has a neighbor that is 0.06 away from it and the span length of the neighboring wire on the other side is 0.08 (<0.09). However, the spacing between the two wires is 0.20, which falls in the forbidden spacing range.



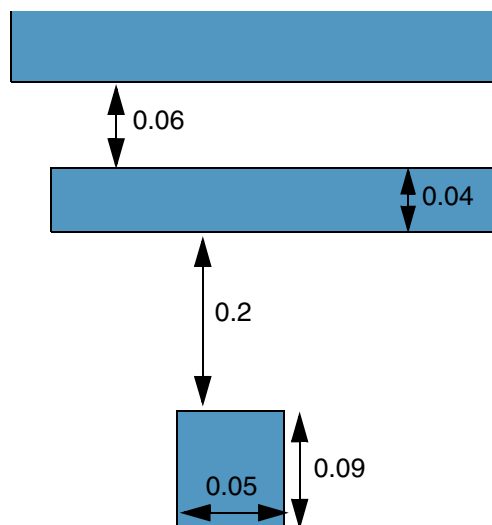
b) The constraint does not apply because the wire with width 0.04 (<0.05) does not have an exact spacing of 0.06 with the neighboring wire (the topmost wire).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply because the span length of the bottommost wire is 0.09 (and not less than 0.09).



d) The constraint does not apply because the span length of the bottommost wire is 0.09 (and not less than 0.09).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

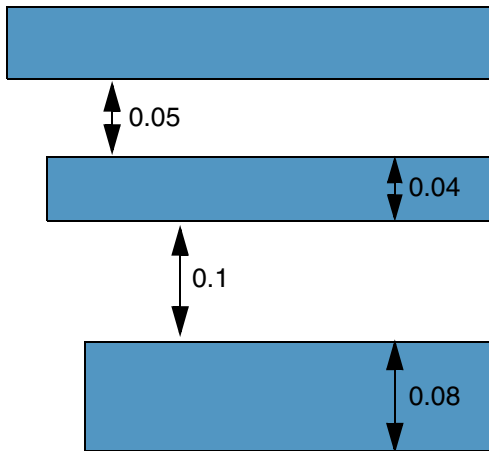
Example 5: forbiddenEdgePitchRange with width, exactSpacingNeighbor, exactWireWidth, and otherWidth

The spacing between two Metal1 wires must not be greater than or equal to 0.08 and less than or equal to 0.12 if the following conditions are met:

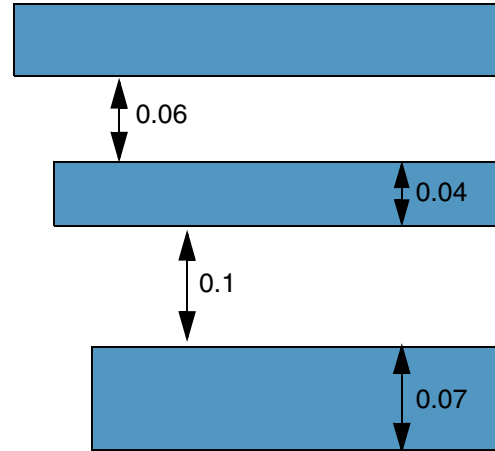
- The width of the first wire is equal to 0.04 and has a neighboring wire at a distance equal to 0.06 from it.
- The width of the second wire is greater than or equal to 0.08.

```
spacings (  
  ( forbiddenEdgePitchRange "Metal1"  
    'width 0.04  
    'exactSpacingNeighbor 0.06  
    'exactWireWidth  
    'otherWidth 0.08  
    "[0.08 0.12]"  
  )  
) ;spacings
```

■ Metal1



a) The constraint does not apply because the topmost wire does not have an exact spacing of 0.06 with the wire with width 0.04.



b) The constraint does not apply because the width of the bottommost wire is 0.07 (and not greater than or equal to 0.08).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

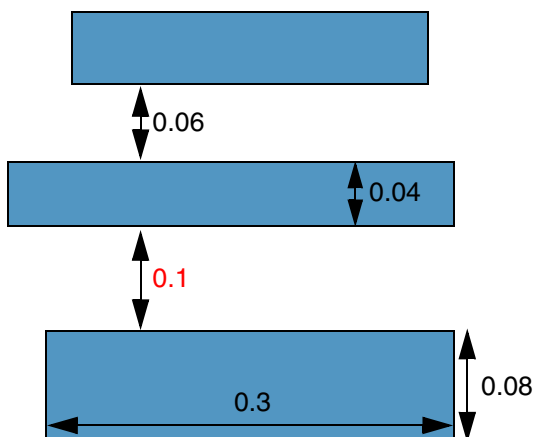
Example 6: forbiddenEdgePitchRange with width, prl, exactSpacingNeighbor, spanLength, and within

The spacing between two Metal1 wires must not be greater than or equal to 0.08 and less than or equal to 0.12 if the following conditions are met:

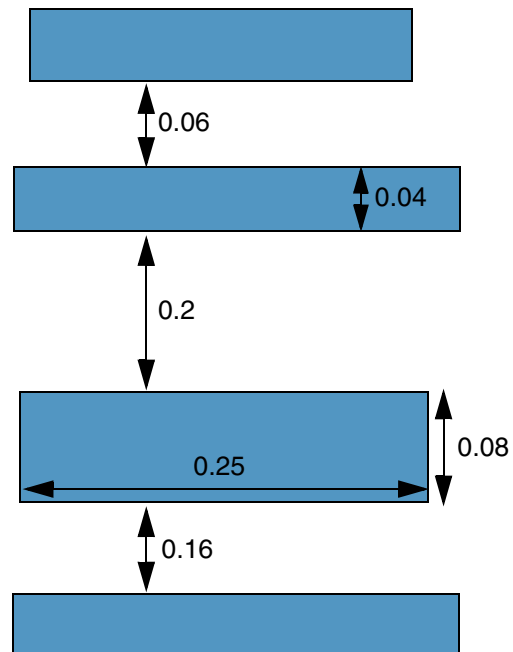
- The first wire is less than 0.05 wide and has a neighboring wire at a distance equal to 0.06 from it.
- The second wire has span length less than 0.09 and has a neighboring wire at a distance less than 0.15 from it.
- The common projected parallel run length between the two wires is greater than 0.04.

```
spacings (  
  ( forbiddenEdgePitchRange "Metal1"  
    'width 0.05 'prl 0.04  
    'exactSpacingNeighbor 0.06  
    'spanLength 0.09 'within 0.15  
    "[0.08 0.12]"  
  )  
) ;spacings
```

■ Metal1



a) FAIL. The wire with span length 0.08 (>0.15) has a neighboring wire at a distance less than 0.15 from it. However, the distance between the middle wire (first wire) and the bottom wire (second wire) is 0.1, which lies in the forbidden spacing range (≥ 0.08 and ≤ 0.12).



b) The constraint does not apply because the wire with span length 0.08 (>0.15) does not have a neighboring wire at a distance less than 0.15 from it.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

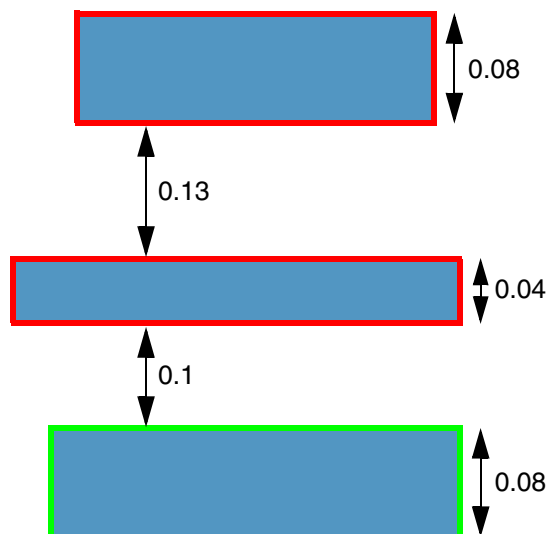
Example 7: forbiddenEdgePitchRange with width, minWidth, outerWidth, outerWithin, sameMask

The spacing between two Metal1 wires on the same mask must not be greater than or equal to 0.08 and less than or equal to 0.12 if the following conditions are met:

- The middle wire is less than 0.05 wide.
- The outer wires are greater than 0.07 wide and the distance between them is less than 0.35.

```
spacings (  
  ( forbiddenEdgePitchRange "Metal1"  
    'width 0.05  
    'minWidth 0  
    'outerWidth 0.07  
    'outerWithin 0.35  
    'sameMask  
    "[0.08 0.12]"  
  )  
) ;spacings
```

■ Metal1
■ Mask1
■ Mask2



PASS. The width of the middle wire is 0.04 (<0.05) and the width of the outer wire that is on the same mask as the middle wire (Mask1) is 0.08 (>0.07). Additionally, the distance between these two wires on the same mask is 0.13, which lies outside the forbidden spacing range (≥ 0.08 and ≤ 0.12). A violation would occur if 'sameMask' was not specified because the distance between the the middle and bottom wires lies in the forbidden spacing range.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

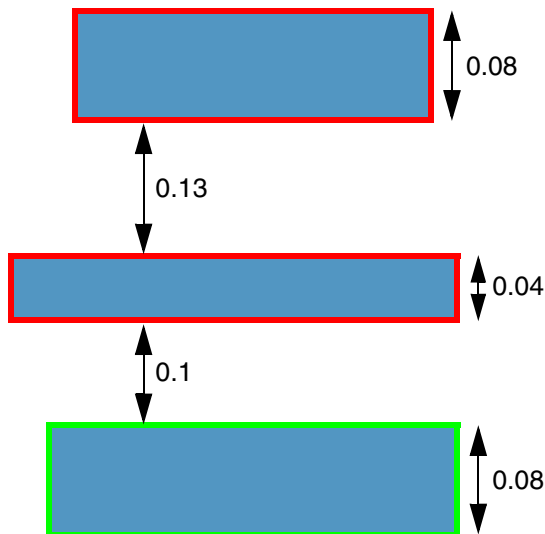
Example 8: forbiddenEdgePitchRange with width, minWidth, outerWidth, outerWithin, outerSameMask

The spacing between two Metal1 wires must not be greater than or equal to 0.08 and less than or equal to 0.12 if the following conditions are met:

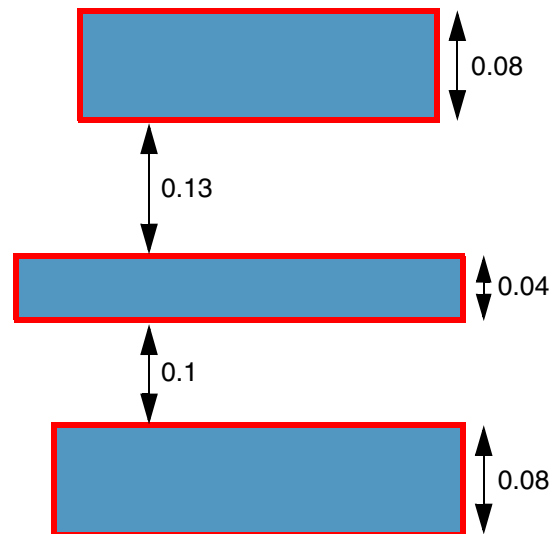
- The middle wire is less than 0.05 wide.
- The outer wires are greater than 0.07 wide and the distance between them is less than 0.35.
- The outer wires are on the same mask.

```
spacings (  
  ( forbiddenEdgePitchRange "Metal1"  
    'width 0.05  
    'minWidth 0  
    'outerWidth 0.07  
    'outerWithin 0.35  
    'outerSameMask  
    "[0.08 0.12]"  
  )  
) ;spacings
```

■ Metal1
■ Mask1
■ Mask2



a) The constraint does not apply because the two outer wires are on different masks. A violation would occur if 'outerSameMask was not specified because the spacing between the middle and bottom wires lies in the forbidden spacing range (≥ 0.08 and ≤ 0.12).



b) FAIL. The spacing between the middle and bottom wires lies in the forbidden spacing range (≥ 0.08 and ≤ 0.12).

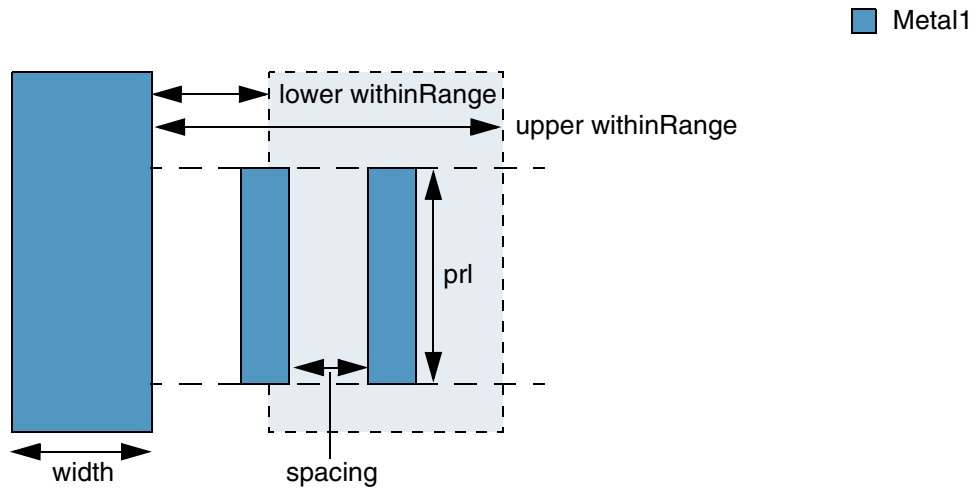
forbiddenProximitySpacing

```

spacings (
    ( forbiddenProximitySpacing tx_layer
      'width f_width
      'prl f_prl
      'withinRange (g_withinRange)
      g_range
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Standard) Restricts the spacing between two shapes on a layer. The constraint applies only if the two shapes are within *withinRange* of a shape with width greater than or equal to *width* and the parallel run length common between the three neighboring shapes is greater than *prl*. A violation occurs if spacing between the two shapes falls in the forbidden spacing range.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_range</i>	The forbidden spacing range, that is, the spacing between edges must not fall in this range. The forbidden spacing range is specified using the format: $[f_{min} \ f_{max}]$ where, $f_{min} \leq \text{spacing} \leq f_{max}$ Type: Floating-point values specifying a <u>range</u> of spacings that are not allowed

Parameters

<i>'width f_width</i>	The constraint applies to shapes that are within a distance specified using <i>withinRange</i> from a shape with width greater than or equal to this value.
<i>'prl f_prl</i>	The constraint applies only if the parallel run length common between the three neighboring shapes is greater than this value.
<i>'withinRange (g_withinRange)</i>	The constraint applies to shapes if their distance from a shape with width greater than or equal to <i>width</i> falls in this range. Type: Floating-point values specifying a <u>range</u> of spacings that trigger the constraint

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

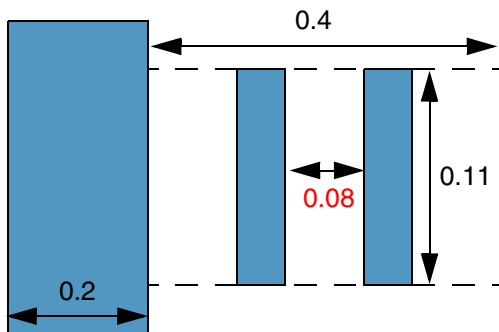
Example

The spacing between two wires must not be greater than or equal to 0.07 and less than or equal to 0.09 if the following conditions are met:

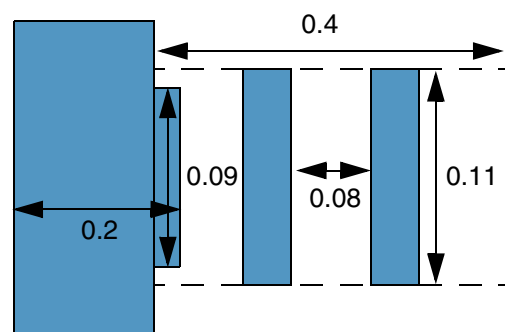
- The two wires are at a distance less than or equal to 0.4 from a wire with width greater than or equal to 0.2.
- The parallel run length common between the three neighboring wires is greater than 0.1.

```
spacings(  
  ( forbiddenProximitySpacing "Metal1"  
    'width 0.2  
    'prl 0.1  
    'withinRange ("<=0.4")  
    ("[0.07 0.09]")  
  )  
) ;spacings
```

■ Metal1



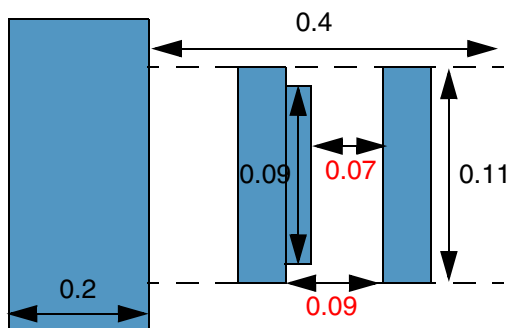
a) FAIL. The two wires are at a distance less than or equal to 0.4 from a wire with width 0.2 and the parallel run length common to the three wires is 0.11 (>0.1). However, the spacing between the two wires is 0.08, which falls in the forbidden range.



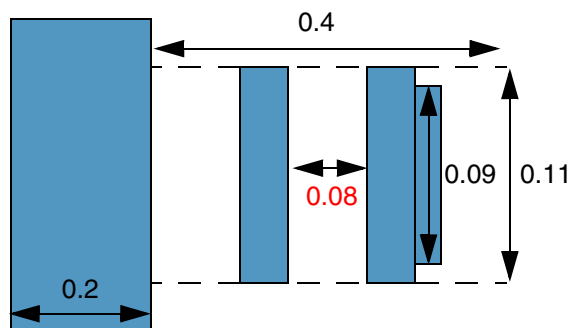
b) The constraint does not apply because the part of the wide wire with width 0.2 has parallel run length of only 0.09 (<0.1) with the neighboring wires.

Virtuoso Technology Data Constraints Reference

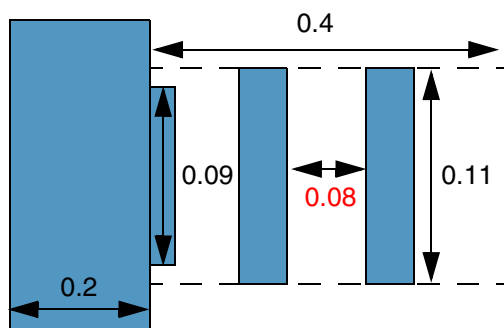
Spacing Constraints (One Layer)



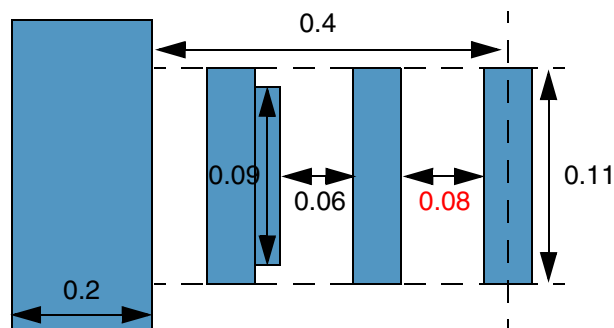
c) FAIL. The two wires are at a distance less than or equal to 0.4 from a wire with width 0.2 and the overall parallel run length common to the three wires is 0.11 (>0.1). However, the spacings of 0.07 at the jog and 0.09 for the rest of the wire fall in the forbidden range.



d) FAIL. The two wires are at a distance less than or equal to 0.4 from a wire with width 0.2 and the parallel run length common to the three wires is 0.11 (>0.1). However, the spacing between the two wires is 0.08, which falls in the forbidden range.



e) FAIL. The two wires are at a distance less than 0.4 from a wire with width 0.2 and the parallel run length common to the three wires is 0.11 (>0.1). However, the spacing between the two wires is 0.08, which falls in the forbidden range.



f) FAIL. The jog with spacing of 0.06 falls outside the forbidden range, but there is a third wire at a distance less than or equal to 0.4 with spacing 0.08, which falls in the forbidden range.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

forbiddenSpacingRanges

```
spacings (
  ( forbiddenSpacingRanges tx_Layer
    ['sameMask | 'mask1 | 'mask2 | 'mask3]
    ['horizontal | 'vertical]
    ['ignoreMiddle]
    'firstSpanLength (f_firstSpan1 f_firstSpan2)
    'secondSpanLength (f_secondSpan1 f_secondSpan2)
    ['firstRectOnly | 'firstExceptRectOnly]
    ['secondRectOnly | 'secondExceptRectOnly]
    ['firstOtherSpanLength g_firstOtherSpanLength
      ['firstExactEqual | 'firstLessThan]
    | 'orthogonalNonEol g_orthogonalNonEol
    | 'secondNonEol (g_minWidth g_maxWidth)]
    ['secondOtherSpanLength g_secondOtherSpanLength
      ['secondExactEqual | 'secondLessThan]
    | 'secondOtherEdgeLength g_secondOtherEdgeLength]
    ['offGridOnly (g_minWidth g_maxWidth)
      ['otherTwoWidth (g_minWidth g_maxWidth)]]
    'prl f_prl
    ['unionPrl g_spanLength]
    (g_ranges)
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies a list of forbidden spacing ranges between two wires on the given layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The spacing that is forbidden between the two wires. Type: A list of floating-point value <u>ranges</u> specifying forbidden spacing

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

'sameMask	The forbidden spacing is applied between two wires on the same mask. If a wire on a different mask is present between the two wires, it is ignored. Type: Boolean
'mask1 'mask2 'mask3	The mask on which forbidden spacing is applied. Type: Boolean
'horizontal 'vertical	The direction in which forbidden spacing is applied. Type: Boolean
'ignoreMiddle	(Virtuoso Layout Suite EXL and higher tiers) The constraint ignores any wires between two wires to which it applies.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'firstSpanLength (*f_firstSpan1* *f_firstSpan2*)

The constraint applies only if one of the two wires has span length greater than or equal to *firstSpan1* and less than or equal to *firstSpan2*.

'secondSpanLength (*f_secondSpan1* *f_secondSpan2*)

The constraint applies only if the other wire has span length greater than or equal to *secondSpan1* and less than or equal to *secondSpan2*.

'firstRectOnly (Virtuoso Layout Suite EXL and higher tiers) The constraint applies on rectangular wires only.

'firstExceptRectOnly

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply on rectangular wires.

'secondRectOnly (Virtuoso Layout Suite EXL and higher tiers) The constraint applies on rectangular wires only.

'secondExceptRectOnly

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply on rectangular wires.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
'firstOtherSpanLength g_firstOtherSpanLength  
|'firstExactEqual  
|'firstLessThan  
|'firstOtherEdgeLength g_firstOtherEdgeLength  
|'orthogonalNonEol g_orthogonalNonEol  
|'secondNonEol (g_minWidth g_maxWidth)
```

The `firstOtherSpanLength` constraint applies only if the span length in the orthogonal direction of `firstSpanLength` is more than the given value.

(Virtuoso Layout Suite EXL and higher tiers) The `firstExactEqual` constraint changes the comparison operation to exactly equal to the value specified in the `firstOtherSpanLength` parameter.

(Virtuoso Layout Suite EXL and higher tiers) The `firstLessThan` constraint changes the comparison operation to less than in the `firstOtherSpanLength` parameter.

(Virtuoso Layout Suite EXL and higher tiers) The `firstOtherEdgeLength` constraint applies when the entire edge length in the orthogonal direction is greater than the specified value.

(Virtuoso Layout Suite EXL and higher tiers) The `orthogonalNonEol` constraint specifies that the edge in the orthogonal direction of forbidden spacing is not an end-of-line (EOL) with width or length less than the specified value.

(Virtuoso Layout Suite EXL and higher tiers) The `secondNonEol` constraint specifies that another edge is a non-EOL with width greater than or equal to the first specified value and less than or equal to the second specified value.

```
'secondOtherSpanLength g_secondOtherSpanLength  
|'secondExactEqual  
|'secondLessThan  
|'secondOtherEdgeLength g_secondOtherEdgeLength
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The `secondOtherSpanLength` constraint applies only if the span length in the orthogonal direction of `secondSpanLength` is more than the given value.

(Virtuoso Layout Suite EXL and higher tiers) The `secondExactEqual` constraint changes the comparison operation to exactly equal to the value specified in the `secondOtherSpanLength` parameter.

(Virtuoso Layout Suite EXL and higher tiers) The `secondLessThan` constraint changes the comparison operation to less than in the `secondOtherSpanLength` parameter.

(Virtuoso Layout Suite EXL and higher tiers) The `secondOtherEdgeLength` constraint applies when the entire edge length in the orthogonal direction is greater than the specified value.

`'offGridOnly (g_minWidth g_maxWidth)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if at least one of the wires is an off-grid wire in the specified range.

`'otherTwoWidth (g_minWidth g_maxWidth)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the other wire width is in the specified range.

`'prl f_prl`

The constraint applies only if the parallel run length between the two wires is greater than this value.

`'unionPr1 g_spanLength`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the parallel run length which is the union sum of the edges with span length greater than or equal to the specified value against the neighbour edges.

Examples

- Example 1: forbiddenSpacingRanges with firstSpanLength and secondSpanLength
- Example 2: forbiddenSpacingRanges with firstOtherSpanLength and secondOtherSpanLength

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

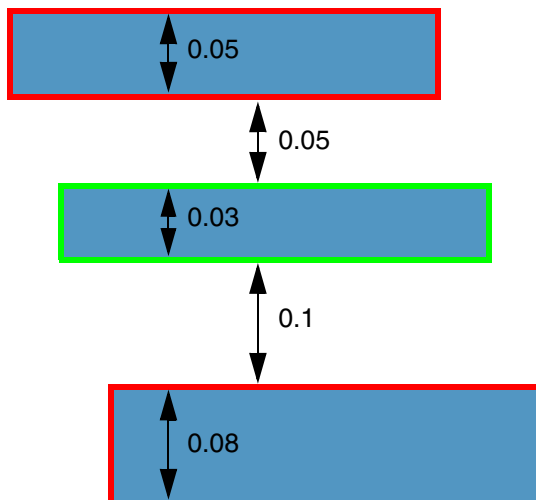
Example 1: forbiddenSpacingRanges with firstSpanLength and secondSpanLength

The spacing between two wires on the same mask must not be greater than or equal to 0.16 and less than or equal to 0.20 if the following conditions are satisfied:

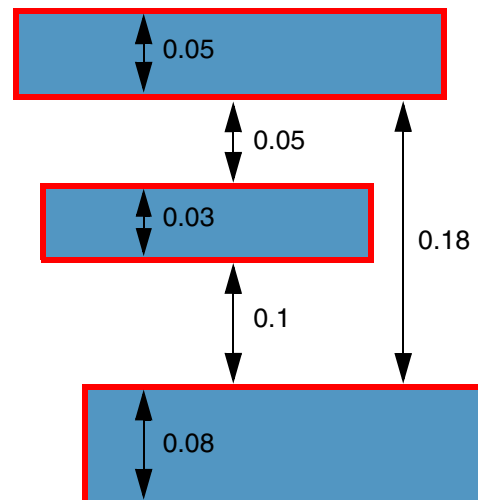
- The width of one of the two wires is greater than or equal to 0.0 and less than or equal to 0.05.
- The width of the second wire is greater than or equal to 0.08 and less than or equal to 3.0.
- The parallel run length between the two wires is greater than 0.

```
spacings (
  ( forbiddenSpacingRanges "Metal1"
    'sameMask
    'firstSpanLength (0.0 0.05)
    'secondSpanLength (0.08 3.0)
    'prl 0
    ("[0.16 0.20]")
  )
) ;spacings
```

■ Metal1
■ mask1
■ mask2



a) FAIL. The spacing between the top and bottom mask1 wires is 0.18, which lies in the forbidden spacing range of "[0.16 0.20]".
If the middle wire was also on mask1, the constraint would apply between the top and middle wire and between the middle and bottom wire, which would result in the constraint being satisfied.



b) FAIL. The distance between the top and bottom mask1 wires toward the right is 0.18, which lies in the forbidden spacing range of "[0.16 0.20]".

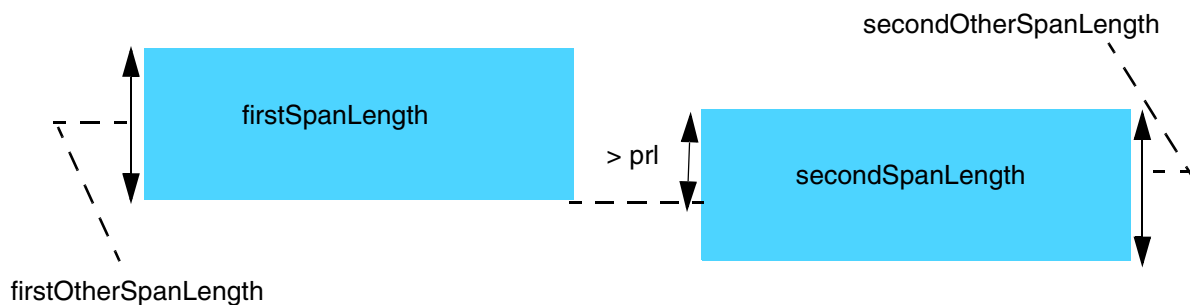
Example 2: forbiddenSpacingRanges with firstOtherSpanLength and secondOtherSpanLength

The horizontal spacing between two wires on the same mask must not be greater than or equal to 0.16 or less than or equal to 0.20 if the following conditions are satisfied:

- The horizontal width of one of the two wires is greater than or equal to 0.0 and less than or equal to 0.05.
- The horizontal width of the second wire is greater than or equal to 0.08 and less than or equal to 3.0.
- The parallel run length between the two wires is greater than 0.
- The span length in the vertical direction of `firstSpanLength` is greater than 0.04.
- The span length in the vertical direction of `secondSpanLength` is greater than 0.06.

```
spacings (
  (forbiddenSpacingRanges "Metal1" 'sameMask 'horizontal
    'firstSpanLength (0.0 0.05)
    'secondSpanLength (0.08 3.0)
    'firstOtherSpanLength 0.04
    'secondOtherSpanLength 0.06
    'prl 0
    ("[0.16 0.20]"))
  )
) ;spacings
```

The following diagram illustrates `firstOtherSpanLength` and `secondOtherSpanLength`:



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

forbiddenEdgePitchRange

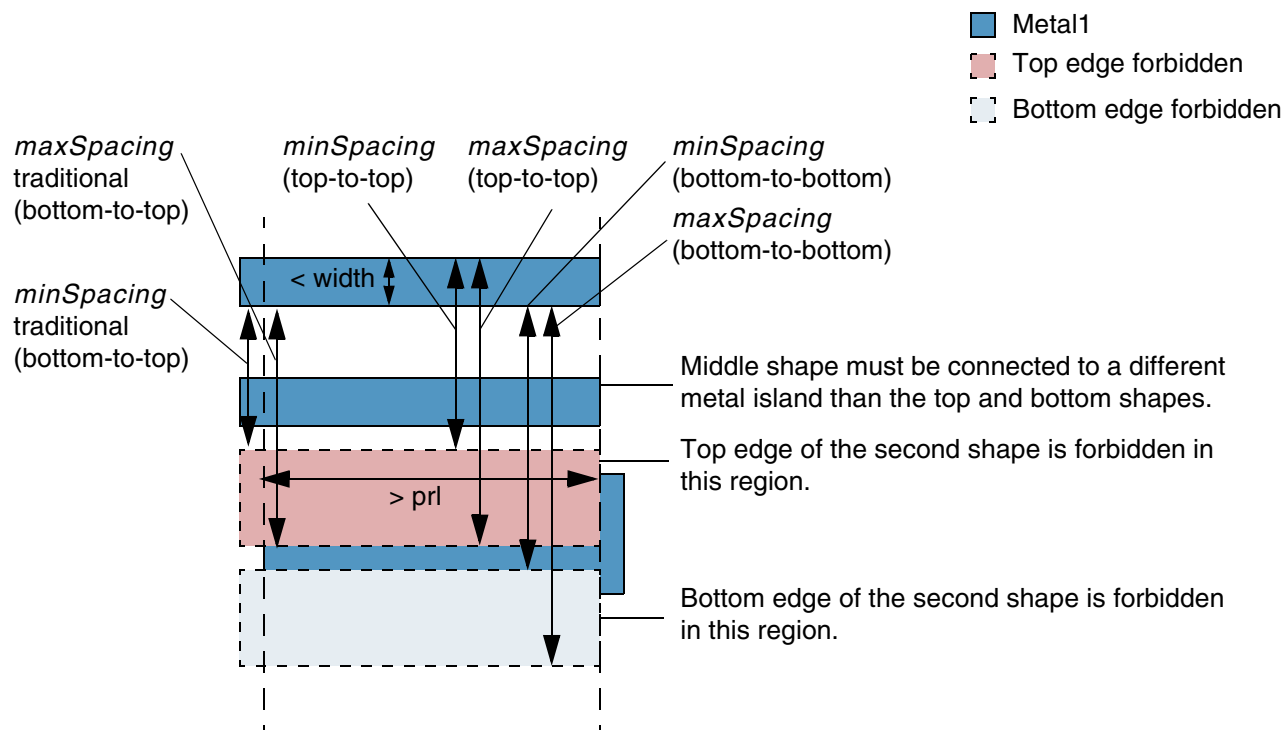
```
spacings (
  ( forbiddenEdgePitchRange tx_layer
    'width f_width ['prl f_prl]
    ['minWidth f_minWidth
      ['outerWidth f_outerWidth]
      ['outerWithin f_outerWithin]
      ['outerSameMask]
    ]
    ['parallelWithin f_parallelWithin
      | 'exactSpacingNeighbor f_exactSpacing
        ['spanLength f_spanLength ['within f_within]]
      | 'neighborSpacingRanges g_neighborSpacingRanges
    ['exceptWireWidthSpacingList (g_exactWidth g_spacing)...]
    ['ignoreMiddleWire (g_Width g_spacing)]
    ]
    ['secondForbiddenRange g_secondForbiddenRange]
    ['sameMask | 'mask1 | 'mask2 | 'mask3]
    ['exactWireWidth]
    ['otherWidth f_otherWidth]
    (g_range)
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Restricts the spacing between the left-to-left and right-to-right edges (or between top-to-top and bottom-to-bottom edges) of two shapes on a layer if a shape from a different metal island lies between the two shapes. The constraint applies only if the width of the first shape is less than the specified value.

Optionally, you can specify that the common projected parallel run length between the two edges and the in-between shape must be greater than a particular value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

g_range

The forbidden spacing range, that is, the spacing between shapes must not fall in this range.

The forbidden spacing range is specified using the format:

$[f\_min1 \ f\_max1]$

where, $f\_min1 \leq \text{spacing} \leq f\_max1$

Type: Floating-point values specifying a range of spacings that are not allowed

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

<code>'width <i>f_width</i></code>	The constraint applies only if the width of the first shape is less than this value.
<code>'minWidth <i>f_minWidth</i></code>	(Virtuoso Advanced Node for Layout Only) The constraint applies to a shape with width less than <i>width</i> and greater than <i>minWidth</i> if the shape is sandwiched by two shapes. The forbidden spacing applies between the middle shape and the outer shapes.
<code>'outerWidth <i>f_outerWidth</i></code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the width of the two outer shapes is greater than this value.
<code>'outerWithin <i>f_outerWithin</i></code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the distance between the two outer shapes is less than this value.
<code>'outerSameMask</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the two outer shapes are on the same mask. The mask applied to the middle shape is ignored. Type: Boolean
<code>'prl <i>f_prl</i></code>	The constraint applies only if the parallel run length between the two edges and the in-between shape is greater than this value.
<code>'parallelWithin <i>f_parallelWithin</i></code>	The constraint applies only if the first shape has neighboring shapes on both sides at a distance less than this value. Note: The spacing between the two shapes is measured from the bottom/top edge to the top/bottom edge (or from the right/left edge to the left/right edge).
<code>'exactSpacingNeighbor <i>f_exactSpacing</i></code>	

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The constraint applies only if the first shape has a neighboring shape at a distance equal to this value on the side opposite to the one on which spacing applies.

Note: The spacing between the two shapes is measured from the bottom/top edge to the top/bottom edge (or from the right/left edge to the left/right edge). A shape from a different metal island need not be present between the two shapes.

`'spanLength f_spanLength`

The constraint applies only if the span length of the second wire is less than this value.

Note: Span length is measured perpendicular to the parallel run length.

`'within f_within`

The constraint applies only if the second shape has a neighboring shape at a distance less than this value from it on either side.

`'neighborSpacingRanges g_neighborSpacingRanges`

The constraint applies only if the spacing between the first shape and a neighboring shape falls in this range.

Type: Floating-point values specifying a range of spacings that trigger the constraint

`'exceptWireWidthSpacingList (g_exactWidth g_spacing)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint specifies that neighbour wires, with at least one of them not in the forbidden spacing range, having width exactly equal to the given value with PRL greater than 0 and the spacing less than the given value are merged as if the gap between the wires is filled. For this parameter, `g_exactWidth` is the width value and `g_spacing` is the spacing value.

`'ignoreMiddleWire (g_width g_spacing)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint specifies that if there is a wire with a width not the same as the value of `g_width` or a spacing to the triggering wire not the same as the value of `g_spacing`, between the triggering wire and the forbidden spacing range wire, that wire is ignored.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'secondForbiddenRange *g_secondForbiddenRange*

A second forbidden spacing range.

The forbidden spacing range is specified using the format:

$[f\_min2 \ f\_max2]$

where, $f\_min2 \leq \text{spacing} \leq f\_max2$

Note: This second forbidden spacing range can be specified only if 'parallelWithin or 'exactSpacingNeighbor is specified.

Type: Floating-point values specifying an additional range of spacings that are not allowed

'sameMask

The constraint applies only to shapes on the same mask.

Note: The shapes between which forbidden spacing is checked must be on the same mask; the mask applied to neighboring shapes is ignored.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'mask1 | 'mask2 | 'mask3

(Virtuoso Advanced Node for Layout Only) The constraint applies only to the shapes on the specified mask.

Type: Boolean

'exactWireWidth

The constraint applies only if the width of the first shape is equal to *width*.

Type: Boolean

'otherWidth *f_otherWidth*

The constraint applies only if the width of the second shape is greater than or equal to this value.

forbiddenSpanLength

```
spacings (
    ( forbiddenSpanLength tx_layer
      {'horizontalEdge | 'verticalEdge}
      'exceptJogLength f_orthogonalSpanLength
      ['treatAsJog]
      g_spanLength
    )
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Defines a span length that is not allowed in the 'edgeDirection parameter unless in a Z or L shape configuration that has a jog length orthogonal to the 'edgeDirection and is equal to the value specified in the f_orthogonalSpanLength parameter.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_spanLength</i>	The span length.

Parameters

'horizontalEdge 'verticalEdge	Specifies the direction of the span length. It is horizontalEdge or verticalEdge.
'exceptJogLength f_orthogonalSpanLength	The constraint applies if the forbidden span length wire is part of a Z shape jog with span length in the orthogonal direction exactly equal to the specified value.
'treatAsJog	The constraint specifies the jog condition applied to L shape.

Example

```
spacings (
    (forbiddenSpanLength "Metal1" 'verticalEdge 'exceptJogLength 0.7
      'treatAsJog
      0.5)
) ;spacings
```

lineEndGap

```
spacings(  
    ( lineEndGap tx_layer  
      'width f_width  
      {'mask1 | 'mask2 | 'mask3}  
      f_spacing  
    )  
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies that two perfectly aligned wires with the specified spacing can be an exemption or a triggering condition on some other constraints that use the `exceptLineEndGap` parameter of the `lineEndGap` constraint.

Values

<i>tx_layer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between two perfectly aligned wires.

Parameters

'width <i>f_width</i>	The constraint applies to perfectly aligned wires that have width less than this value.
'mask1 'mask2 'mask3	The constraint applies to wires on the specified mask.

Example

The constraint applies to two perfectly aligned wires on mask1 when their width is less than 0.09 and they have a spacing of 0.05 between them.

```
spacings(  
    (lineEndGap "Metal1"  
      'width 0.09  
      'mask1  
      0.05  
    )  
); spacings
```


Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

lithoMacroHalo

```
spacings (
    ( lithoMacroHalo tx_layer
      'wireSpacing f_wireSpacing
      ['noSameMetalJog]
      (f_horizontalHalo f_verticalHalo)
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies a routing halo for litho purpose on a block macro. Only planar connections to macro pins are allowed inside this routing halo. Any other wires or via insertions above or below are not allowed.

Values

tx_layer The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_horizontalHalo f_verticalHalo

The halo values in the horizontal and vertical direction. These are exact values that describe the size of the checking region around the block macro.

Parameters

'wireSpacing *f_wireSpacing*

The wires connected to the macro pins must have spacing greater than or equal to this value.

'noSameMetalJog (Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that same layer and same metal jogs and layer change are not allowed in the halo. However, rectangular through wires are allowed.

Example

The routing halo on a block macro must extend 0.1 in the horizontal direction and 0.3 in the vertical direction. The wires connected to the macro pins must have spacing greater than or

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

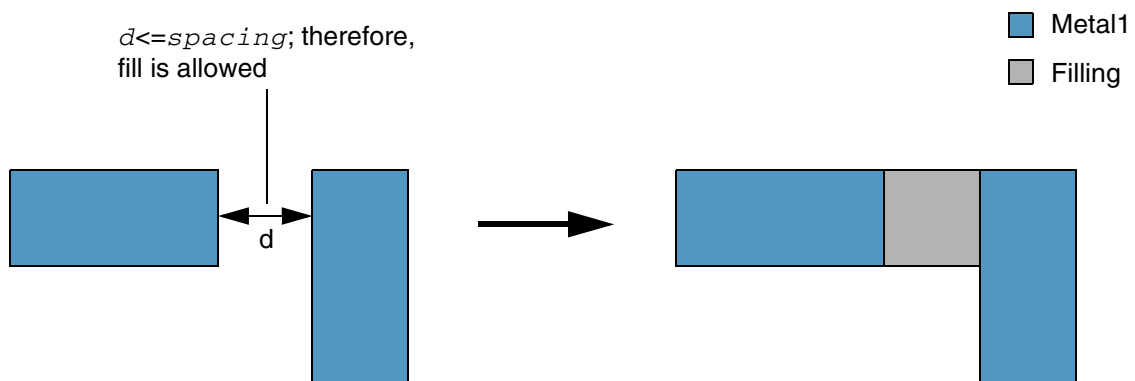
equal to 0.05. Also, same layer and same metal jogs and layer change are not allowed in the halo.

```
spacings (
  ( lithoMacroHalo "Metall1"
    'wireSpacing 0.05
    'noSameMetalJog
    (0.1 0.3)
  )
) ;spacings
```

maxFilling

```
spacings (
    ( maxFilling tx_layer
      f_spacing
    )
) ;spacings
```

Specifies the maximum edge-to-edge space that can be filled between shapes on the specified layer. This constraint is typically used for wells or to fill notches.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The space between shapes must be less than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

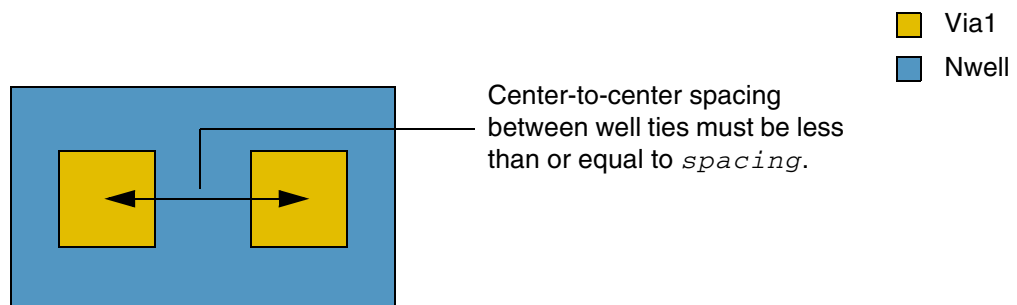
Fill is allowed if the spacing between shapes on `Metal3` is less than or equal to 1.5.

```
spacings(  
    ( maxFilling "Metal3"  
      1.5  
    )  
); spacings
```

maxTapSpacing

```
spacings (
    ( maxTapSpacing tx_layer
      f_spacing
    )
) ;spacings
```

Specifies the maximum center-to-center spacing between two well ties (taps) inside a well on the specified layer. This constraint applies to cut or local interconnect layers.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. The layer must be a cut or local interconnect layer. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The center-to-center spacing between well ties must be less than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The center-to-center spacing between two well ties inside a well on layer Nwell must be less than or equal to 1.5.

```
spacings (
  ( maxTapSpacing "Nwell"
    1.5
  )
) ;spacings
```

minAdjacentFourViaSpacing

```
orderedSpacings (
  ( minAdjacentFourViaSpacing tx_metalLayer tx_cutLayer
    ['cutClass {f_width | (f_width f_length) | t_name}]
    'prl f_parallelRunLength 'within f_within
    ['toAll]
    ['horizontal | 'vertical]
    ['mergedCuts]
    ['exceptDiffMetal ['noExtraCut]]
    ['minCorner | 'minInnerCorner | 'maxInnerCorner]
    ['exceptOverlap g_overlap]
    ['exceptMinLength g_minLength]
    ['cutWithin g_cutWithin]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['exceptOtherWidthWithCut g_width
      'otherLayer g_otherLayer]
    ['width g_width]
    f_spacing
  )
) ;orderedSpacings
```

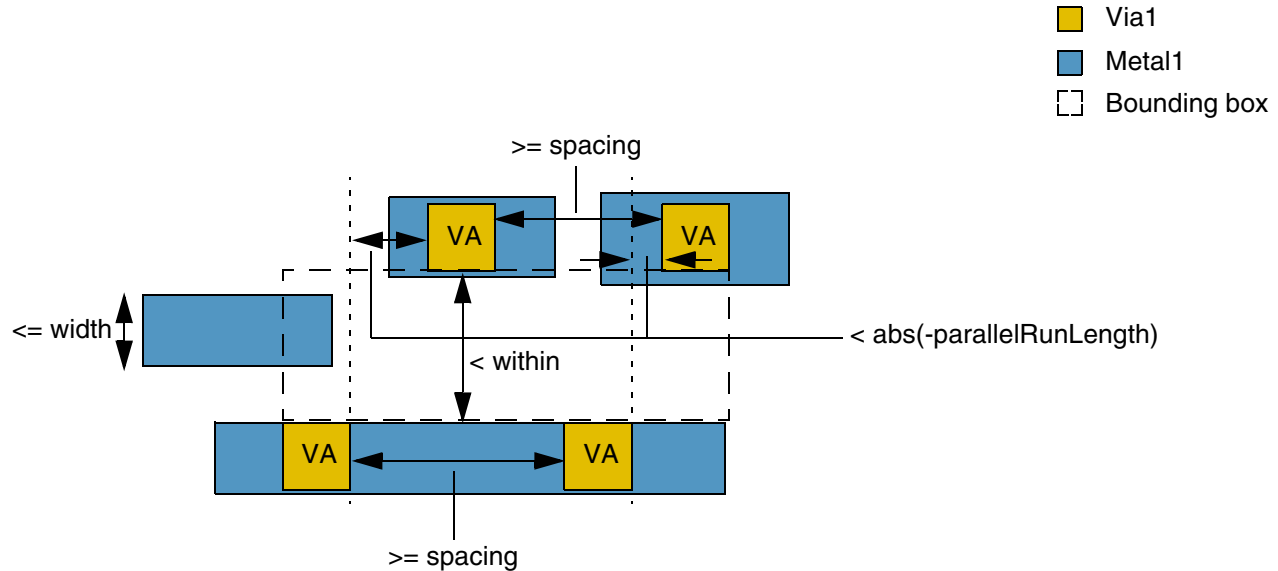
(Virtuoso Advanced Node for Layout Only) Specifies that the spacing between two adjacent via cuts on the same or on different nets must be greater than or equal to *spacing* if the following conditions are met:

- Each via cut has a neighboring via cut at a distance less than *within* from it and has parallel run length greater than *parallelRunLength* with this neighboring via cut.
- A wire on *metalLayer* overlaps the bounding box formed by the via cut edges that face each other.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- The neighbor metal width is less than or equal to `width`.



Values

<code>tx_metalLayer</code>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>tx_cutLayer</code>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>f_spacing</code>	The spacing between the adjacent via cuts must be greater than or equal to this value.

Parameters

```
'cutClass {f_width | (f_width f_length) | t_name}
```


Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class. If not specified, any cut combination of four cuts can trigger the rule.

`'prl f_parallelRunLength`

The constraint applies only if the parallel run length between neighboring via cuts that are less than *within* apart is greater than this value.

`'toAll`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies when at least one cut is of the specified cut class even if the other cuts are of different cut classes.

`'within f_within`

The constraint applies only if the distance between each pair of neighboring via cuts is less than this value.

`'horizontal | 'vertical`

The *spacing* and *parallelRunLength* values are measured in this direction, and the *within* value is measured in the orthogonal direction.

Type: Boolean

`'mergedCuts`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if merged cuts have a parallel run length more than the *prl* value and less than the *within* value.

`'exceptDiffMetal`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if at least one set of two cuts has the same metal on the specified metal layer.

`'noExtraCut`

(Virtuoso Layout Suite EXL and higher tiers) The constraint specifies that the `'exceptDiffMetal` exemption applies only when there is no cut between the four cuts triggering the rule.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'minCorner | 'minInnerCorner | 'maxInnerCorner

(Virtuoso Layout Suite EXL and higher tiers) If neighbor cuts, which are not in the same set, have *prl* greater than 0, the cut center is used by default to form the metal search window, unless one of these parameters are specified:

- **minCorner:** The inner corner is used to form the search window.
- **minInnerCorner:** The minimum bounding box of the inner cut corners is used.
- **maxInnerCorner:** The maximum bounding box of the inner cut corners is used.

'exceptOverlap *g_overlap*

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply when the middle metal neighbor has a parallel run length overlapping the window by less than this value.

'exceptMinLength *g_minLength*

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply when the middle metal neighbor has at least this length.

'cutWithin *g_cutWithin*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies even if the middle metal neighbor does not overlap with the metal search window when the wire contains an above or below cut within the given value of the window.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'exceptOtherWidthWithCut g_width`

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply to a wire on *tx_otherLayer* with a cut on *tx_cutLayer*, when the width of the wire is greater than or equal to *g_width* and it is within the search window formed by the minimum bounding box of the inner cut corners.

`'otherLayer g_otherLayer`

(Virtuoso Layout Suite EXL and higher tiers) The layer name for the wire fulfilling the parameter condition in *g_width*.

`'width g_width`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the neighbor metal width is less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The spacing between two adjacent via cuts must be greater than or equal to 0.07 if the following conditions are met:

- Each via cut has a neighboring via cut at a distance less than 0.06 from it and has parallel run length greater than 0.05 with this neighboring via cut.
- A wire on Metal3 overlaps the bounding box formed by the via cut edges that face each other.
- A wire on Metal2 with a cut on Via2 has width greater than or equal to 0.05 and it is within the search window formed by the minimum bounding box of the inner cut corners.

```
orderedSpacings (
  ( minAdjacentFourViaSpacing "Metal3" "Via2"
    'cutClass "VA"
    'prl 0.05
    'within 0.06
    'exceptOtherWidthWithCut 0.05 'otherLayer "Metal2"
    0.07
  )
) ;orderedSpacings
```

minAdjacentTrimSpacing

```
spacings (
    ( minAdjacentTrimSpacing tx_layer
      'distance f_distance
      ['sameMask] ['mask1 | 'mask2 | 'mask3]
      g_trims
    )
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Defines the maximum number of trim shapes allowed within a specified distance. of each other on adjacent tracks.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_trims</i>	The maximum number of trim shapes on adjacent tracks with spacing less than <i>f_distance</i> .

Parameters

'distance <i>f_distance</i>	The minimum spacing between trims on adjacent trim tracks. The trims are in a staggered pattern on two adjacent trim tracks.
'sameMask	The constraint applies only to trims on the same mask. Type: Boolean
'mask1 'mask2 'mask3	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies to specified trim mask, <i>mask1</i> , <i>mask2</i> , or <i>mask3</i> . Type: Boolean

Virtuoso Technology Data Constraints Reference

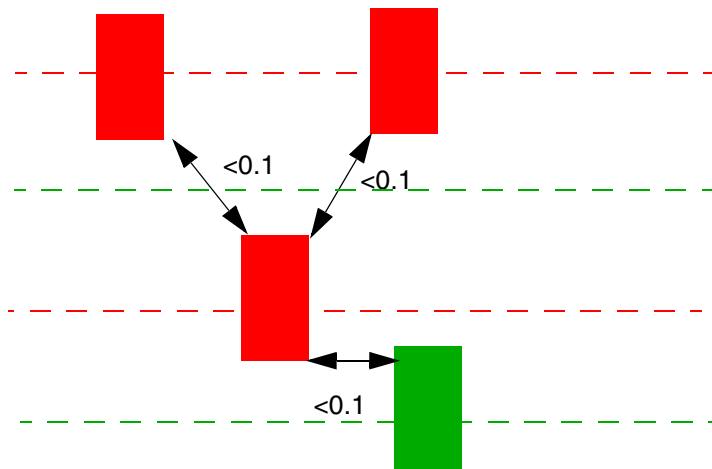
Spacing Constraints (One Layer)

Example

In this example, it is a violation if three or more trim shapes are on adjacent tracks within 0.1 micron of each other.

```
spacings (
  ( minAdjacentTrimSpacing "TrimMetal1"
    'distance 0.1
    'sameMask
    3)
) ;spacings
```

 TrimMetal1 mask1
 TrimMetal1 mask2
- - - Routing tracks for
- - - TrimMetal1 shapes

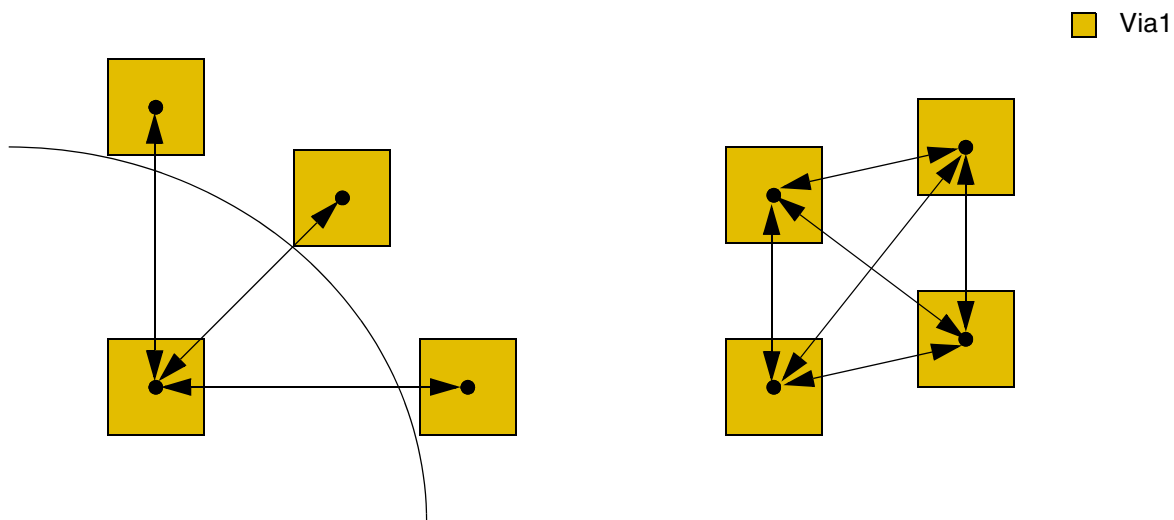


FAIL. There are three trim shapes on mask1 within 0.1 micron. Without 'sameMask' it would be a PASS because adjacent trim tracks would be adjacent metal tracks and there are only one mask1 and one mask2 trim shapes on adjacent tracks.

minCenterToCenterSpacing

```
spacings (
    ( minCenterToCenterSpacing tx_cutLayer
      f_spacing
    )
) ;spacings
```

Specifies the minimum center-to-center spacing between shapes on a cut layer. In particular, this constraint is used to specify the spacing between adjacent via cuts.



Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The center-to-center spacing between cut shapes must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The following center-to-center spacing must be satisfied between via cuts on a layer:

- On layer Via1, center-to-center spacing must be at least 2.0.
- On layer Via2, center-to-center spacing must be at least equal to the value of the technology parameter `mincenterspace1`.

```
spacings (  
  ( minCenterToCenterSpacing "Via1"  
    2.0  
  )  
  ( minCenterToCenterSpacing "Via2"  
    techParam("mincenterspace1")  
  )  
) ;spacings
```


minClusterSpacing (One layer)

```
spacings (
  ( minClusterSpacing tx_layer
    'widthRange g_widthRange
    'numShapesRange g_numShapesRange
    'spacingRange g_spacingRange
    ['centerLineDistance]
    ['minSpaceOverride]
    ['horizontal | 'vertical]
    ['otherWidthRange g_otherWidthRange]
    ['groupHorizontal | 'groupVertical
      ['enclosingLayer tx_enclosingLayer
        ['viaEdgeType x_edgeType]
      ]
    ]
    ['prl f_prl]
    ['clusterToCluster]
    ['edgeToEdge]
    ['sameMask]
    ['widthHorizontal | 'widthVertical]
    f_spacing
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between two neighboring clusters or a cluster and other neighboring shapes that may or may not be part of another cluster, on the same layer. The shapes in a cluster must be exactly aligned and must satisfy all of the following conditions:

- The width of each shape in the cluster must be in the specified width range, *widthRange*.
- The edge-to-edge spacing or the spacing between the centerlines of neighboring shapes in the cluster must be in the specified spacing range, *spacingRange*.
- The number of shapes that satisfy the two conditions listed above must be in the specified number range, *numShapesRange*.

Optionally, each cluster can be aligned either in the vertical direction or in the horizontal direction. The constraint can also be selectively applied to neighboring non-cluster shapes based on their widths.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between a cluster and a neighboring shape must be greater than or equal to this value. 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.

Parameters

<i>'widthRange g_widthRange</i>	The widths of the shapes in a cluster must fall in this range. Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint
<i>'numShapesRange g_numShapesRange</i>	The constraint applies only if the number of shapes in a cluster falls in this range. Type: Integer values specifying the minimum and maximum number of shapes (<u>range</u>) that a cluster can contain
<i>'spacingRange g_spacingRange</i>	This spacing between the centerlines of the shapes in a cluster must fall in this range. Type: Floating-point values specifying a <u>range</u> of spacings that trigger the constraint
<i>'centerLineDistance</i>	The spacing is measured from the centerline of the cluster to centerline of the neighboring non-cluster shapes. Note: The intra-cluster spacing is always measured from the centerline, irrespective of whether this parameter is specified. Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'minSpaceOverride`

The constraint does not apply if the neighboring shape is part of another cluster.

Note: The `minSpacing` constraint applies instead.

Type: Boolean

`'horizontal | 'vertical`

The direction in which spacing is measured. If direction is not specified, spacing is measured in any direction.

Type: Boolean

`'otherWidthRange g_otherWidthRange`

The constraint applies to a non-cluster shape only if its width falls in this range.

Type: Floating-point values specifying a range of widths that trigger the constraint

`'groupHorizontal | 'groupVertical`

The constraint applies only if a cluster is formed in the specified direction. If the direction is not specified, the cluster can be formed in either direction.

Type: Boolean

`'enclosingLayer tx_enclosingLayer`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only for the specified group direction for shapes enclosed by this layer.

`'viaEdgeType x_edgeType`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when all via edges in the direction orthogonal to the group direction and in the enclosing layer are of this type. The type is defined in the `viaEdgeType` constraint.

`'prl f_prl`

The constraint applies only if the parallel run length between the shapes in a cluster and a neighboring non-cluster shape is greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

<code>'clusterToCluster</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only between neighboring clusters. Otherwise, the constraint also applies between a cluster and neighboring non-cluster shapes. Type: Boolean
<code>'edgeToEdge</code>	(Virtuoso Advanced Node for Layout Only) 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 <code>'centerLineDistance</code> parameter determines how spacing is measured between a cluster and neighboring non-cluster shapes. Type: Boolean
<code>'sameMask</code>	(Virtuoso Advanced Node for Layout Only) Clusters are formed between shapes of the same color. Moreover, the constraint applies to neighboring non-cluster shapes only if they are of the same color as the shapes that form the cluster. Type: Boolean
<code>'widthHorizontal</code> <code>'widthVertical</code>	(Virtuoso Advanced Node for Layout Only) The direction in which the <code>widthRange</code> and <code>otherWidthRange</code> parameters are measured. These parameters would be always measured in this direction, even if it does not correspond to the smaller dimension. By default, when this parameter is not used, the smaller dimension is used for <code>widthRange</code> and <code>otherWidthRange</code>. Type: Boolean

Examples

- Example 1: minClusterSpacing with widthRange, numShapesRange, spacingRange, and otherWidthRange
- Example 2: minClusterSpacing with widthRange, numShapesRange, spacingRange, centerLineDistance, and otherWidthRange

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- Example 3: minClusterSpacing with widthRange, numShapesRange, spacingRange, centerLineDistance, otherWidthRange, and groupVertical
- Example 4: minClusterSpacing with widthRange, numShapesRange, spacingRange, horizontal, otherWidthRange, and groupVertical
- Example 5: minClusterSpacing with widthRange, numShapesRange, spacingRange, minSpaceOverride, otherWidthRange, and prl
- Example 6: minClusterSpacing with widthRange, numShapesRange, spacingRange, and clusterToCluster
- Example 7: minClusterSpacing with widthRange, numShapesRange, spacingRange, otherWidthRange, and sameMask
- Example 8: minClusterSpacing with group direction, enclosingLayer, and viaEdgeType

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 1: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, and *otherWidthRange*

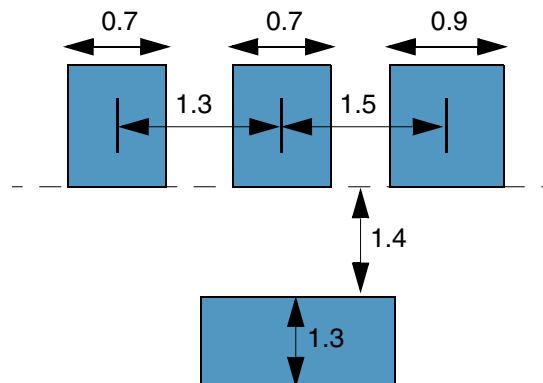
The spacing between a cluster and a neighboring shape must be at least 1.4 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5.

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
spacings (  
  ( minClusterSpacing "Metal1"  
    'widthRange "[0.7 0.9]"  
    'numShapesRange "[3 4]"  
    'spacingRange "[1.3 1.5]"  
    'otherWidthRange "[1.2 1.5]"  
    1.4  
  )  
) ;spacings
```

■ Metal1



PASS. All conditions for a cluster are satisfied: the number of shapes is 3 (which falls in the range [3 4]); the widths of the shapes fall in the range [0.7 0.9]; and the spacing measured between centerlines falls in the range [1.3 1.5]. The width of the non-cluster shape is 1.3, which falls in the range [1.2 1.5], and the spacing between the shapes in the cluster and the non-cluster shape is 1.4.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 2: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *centerLineDistance*, and *otherWidthRange*

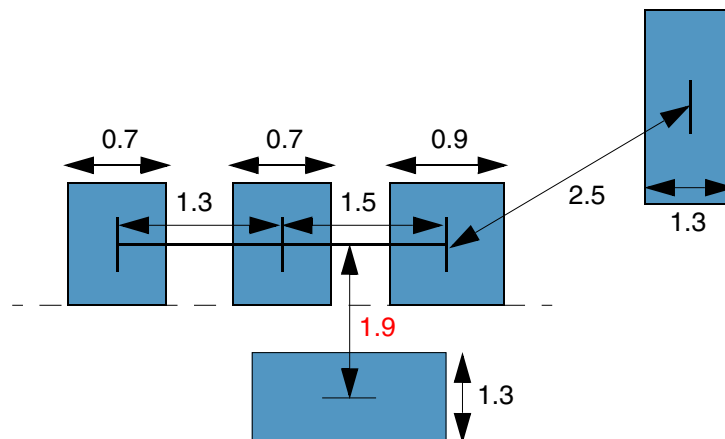
The spacing measured from the centerline of a cluster to the centerline of a neighboring shape must be at least 2.0 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5.

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
spacings (  
  ( minClusterSpacing "Metal1"  
    'widthRange "[0.7 0.9]"  
    'numShapesRange "[3 4]"  
    'spacingRange "[1.3 1.5]"  
    'centerLineDistance  
    'otherWidthRange "[1.2 1.5]"  
    2.0  
  )  
) ;spacings
```

■ Metal1



FAIL. All conditions for a cluster are satisfied and the width of the two neighboring shapes is 1.3 (falls in [1.2 1.5]). The spacing between the cluster centerline and the centerline of the neighboring shape on the right is 2.5 (>2.0), but the spacing between the cluster centerline and the centerline of the neighboring shape at the bottom is 1.9 (<2.0).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 3: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *centerLineDistance*, *otherWidthRange*, and *groupVertical*

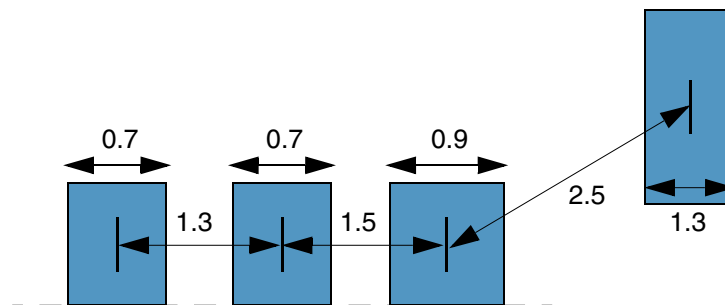
The spacing measured from the centerline of a vertical cluster to the centerline of a neighboring shape must be at least 2.0 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5.

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
spacings (  
  ( minClusterSpacing "Metal1"  
    'widthRange "[0.7 0.9]"  
    'numShapesRange "[3 4]"  
    'spacingRange "[1.3 1.5]"  
    'centerLineDistance  
    'otherWidthRange "[1.2 1.5]"  
    'groupVertical  
      3.0  
  )  
) ;spacings
```

■ Metal1



The constraint does not apply because the cluster direction is horizontal.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

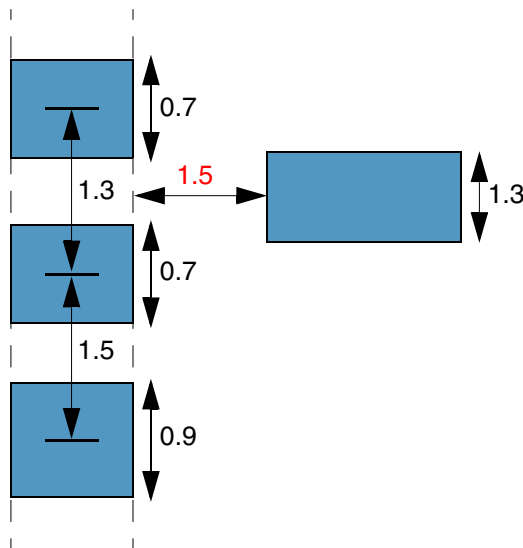
Example 4: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *horizontal*, *otherWidthRange*, and *groupVertical*

The horizontal spacing between a vertical cluster and a neighboring shape must be at least 1.7 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5. The shapes in the cluster must satisfy the following conditions:

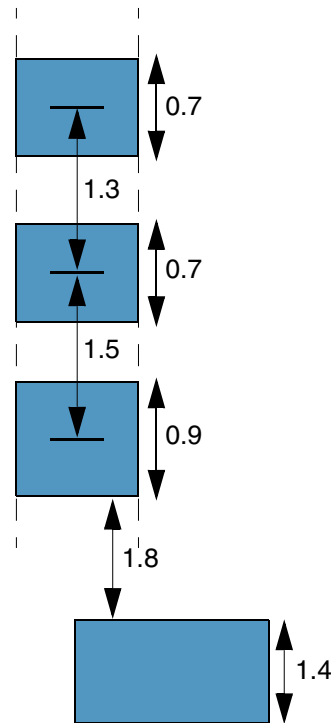
- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
spacings (
  ( minClusterSpacing "Metal1"
    'widthRange "[0.7 0.9]"
    'numShapesRange "[3 4]"
    'spacingRange "[1.3 1.5]"
    'horizontal
    'otherWidthRange "[1.2 1.5]"
    'groupVertical
    1.7
  )
) ;spacings
```

■ Metal1



a) FAIL. All conditions for a cluster are satisfied, the cluster direction is vertical, and the width of the non-cluster shape is 1.3 (falls in [1.2 1.5]). Additionally, the spacing between the shapes in the cluster and the neighboring non-cluster shape is measured in the horizontal direction. However, the spacing is only 1.5 (<1.7).



b) The constraint does not apply because the spacing between the shapes in the cluster and the neighboring non-cluster shape is measured in the vertical direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 5: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *minSpaceOverride*, *otherWidthRange*, and *prl*

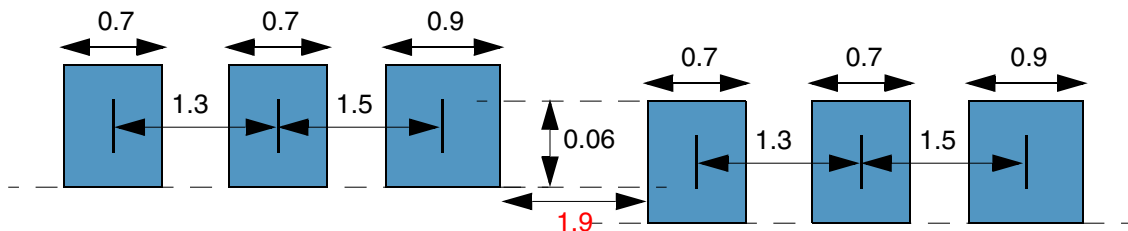
The spacing between a cluster and a neighboring shape must be at least 1.4 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5 and the common parallel run length is greater than or equal to 0.05.

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

If the neighboring shape belongs to a cluster, a minimum spacing requirement of 2.0 must be satisfied.

```
spacings(                                     ■ Metal1
  ( minSpacing "Metal1" 2.0 )
  ( minClusterSpacing "Metal1"
    'widthRange "[0.7 0.9]"
    'numShapesRange "[3 4]"
    'spacingRange "[1.3 1.5]"
    'minSpaceOverride
    'otherWidthRange "[1.2 1.5]"
    'prl 0.05
    1.4
  )
) ;spacings
```



FAIL. The group of shapes on the left satisfies all conditions for a cluster and shares a parallel run length of 0.06 (>0.05) with a neighboring shape that is part of another cluster. Therefore, the `minClusterSpacing` constraint does not apply (because `'minSpaceOverride` is specified). Instead, the `minSpacing` constraint applies and requires a spacing of 2.0 between shapes, a condition that is not satisfied. If `'minSpaceOverride` was not specified, the constraint would pass because the spacing between clusters is 1.9 (>1.4).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 6: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, and *clusterToCluster*

The spacing between two clusters must be at least *minS* if the shapes in both clusters satisfy the following conditions:

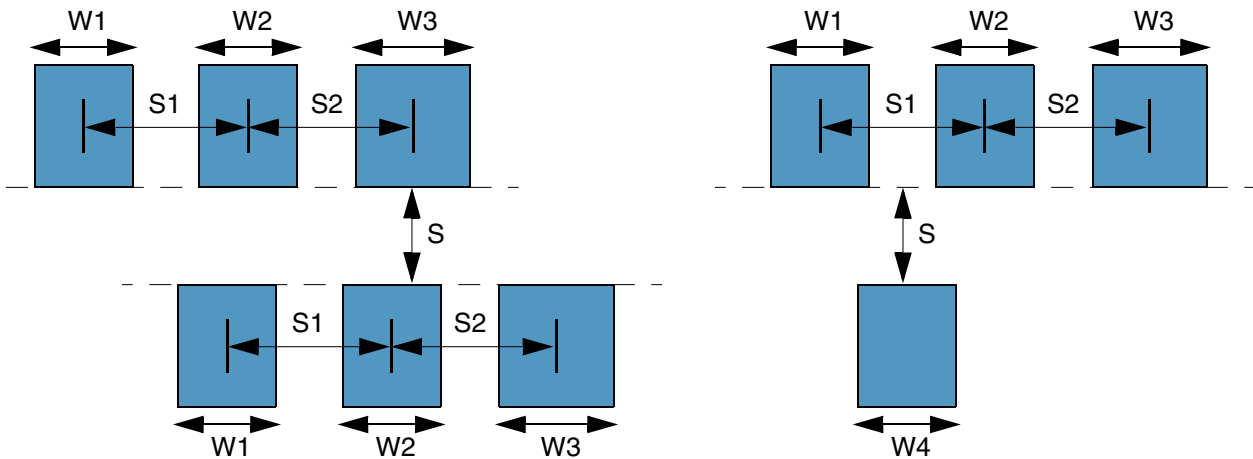
- The width of each shape in both clusters is in the range *W*.
- The number of shapes in both clusters is in the range *NR*.
- The spacing measured between the centerlines of neighboring shapes in both clusters is in the range *SR*.

```

spacings (
  ( minClusterSpacing "Metal1"
    'widthRange W
    'numShapesRange NR
    'spacingRange SR
    'clusterToCluster
    minS
  )
) ;spacings

```

■ Metal1



a) The constraint applies if $W1$, $W2$, and $W3$ fall in the range W and the number of shapes (3) falls in the range NR . The constraint is violated if $S < minS$.

b) The constraint does not apply. When 'clusterToCluster' is specified, the constraint applies only between clusters.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 7: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *otherWidthRange*, and *sameMask*

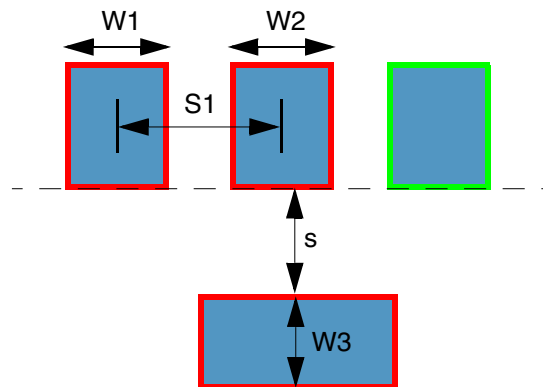
The spacing between a cluster of shapes on mask1 and a neighboring mask1 shape must be at least $minS$ if the width of the neighboring shape falls in the range WO .

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range W .
- The number of shapes in the cluster must be in the range NR .
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range SR .

```
spacings (  
  ( minClusterSpacing "Metal1"  
    'widthRange W  
    'numShapesRange NR  
    'spacingRange SR  
    'otherWidthRange WO  
    'sameMask  
    minS  
  )  
) ;spacings
```

■ Metal1
■ Metal1, mask1
■ Metal1, mask2



The constraint applies if $W1$ and $W2$ fall in the range W ; $W3$ falls in the range WO ; and the number of shapes (2) falls in the range NR . The constraint is violated if $S < minS$.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

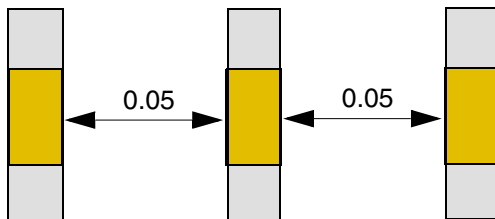
Example 8: *minClusterSpacing* with *group direction*, *enclosingLayer*, and *viaEdgeType*

The spacing between a cluster of shapes on mask1 and a neighboring mask1 shape must be as defined in the constraint. There must be only two self-aligned vias with self-aligned edges facing each other within a distance of 0.04 to 0.108.

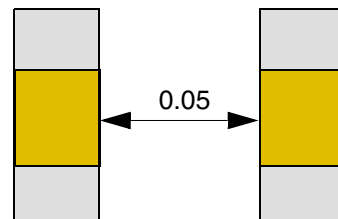
```

spacings (
  (viaEdgeType "V5"
    'anyOppositeExtension 0
    'edgeExtension .001
    1
  )
  ( minClusterSpacing "V5"
    'widthRange "[.03 .05]"
    'numShapesRange ">=3"
    'spacingRange "[0.02 0.09]"
    'groupHorizontal
    'edgeToEdge
    'enclosingLayer "M6"
    'viaEdgeType 1
    0
  )
) ;spacings

```



FAIL. The constraint applies because all edges in the direction perpendicular to the cluster direction have extensions equal to 0. There are 3 shapes in the cluster, which violates the `numShapesRange` parameter.



PASS. The constraint does not apply because there are only 2 shapes in the cluster.



The constraint does not apply because the extensions in the direction perpendicular to the cluster direction are equal to 0.

minConcaveCornerSpacing

```
spacingTables (
  ( minConcaveCornerSpacing tx_layer
    (( "width" nil nil )
      ['minLength f_minLength]
      ['exceptNotch | 'exceptNotchLength f_notchLength]
    )
    (g_table)
  )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Only) Specifies a pair of spacing values that define a forbidden region on a concave corner, horizontally by the first given value and vertically by the second given value, for any neighboring wire with width greater than or equal to the given width value.

Values

tx_layer The layer on which the constraint is applied.
Type: String (layer and purpose names) or Integer (layer number)

"width" nil nil
This identifies the index for *table*.

g_table The format of the *table* row is as follows:
f_width ((*f_horiSpacing* *f_vertSpacing*))
where,

- *f_width* is the width of the neighboring wire. The corresponding spacing values apply if the width of the neighboring wire is greater than or equal to this value.
- *f_horiSpacing* is dimension of the forbidden region in the horizontal direction.
- *f_vertSpacing* is dimension of the forbidden region in the vertical direction.

Type: A 1-D table specifying a pair of floating-point spacing values, indexed on the width of the neighboring wire

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

'minLength *f_minLength*

The constraint applies only if the lengths of the concave edges are greater than or equal to this value.

'exceptNotch

The constraint does not apply to a U-shaped notch if the notch length is less than the minimum of the two spacing values, *horiSpacing* and *vertSpacing*.

Type: Boolean

'exceptNotchLength *f_notchLength*

(Virtuoso Advanced Node for Layout Only) The constraint does not apply to a U-shaped notch if the notch length is less than this value.

Examples

- Example 1: minConcaveCornerSpacing with exceptNotch
- Example 2: minConcaveCornerSpacing with exceptNotchLength

Virtuoso Technology Data Constraints Reference

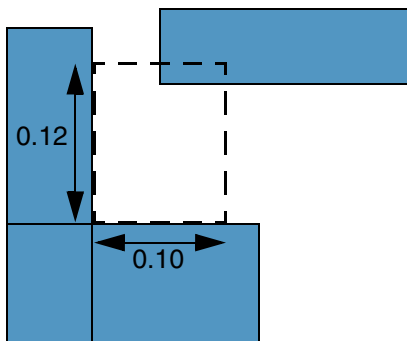
Spacing Constraints (One Layer)

Example 1: *minConcaveCornerSpacing* with *exceptNotch*

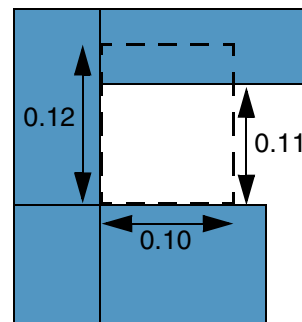
A neighboring wire must not be present inside a forbidden region that extends 0.1 horizontally and 0.12 vertically from a concave corner. The constraint does not apply to a U-shaped notch if the notch length is less than 0.1 (the smaller of the two spacing values).

```
spacingTables (
  ( minConcaveCornerSpacing "Metal1"
    (( "width" nil nil )
      'exceptNotch
    )
    (
      0.0 ((0.1 0.12))
    )
  )
) ;spacingTables
```

■ Metal1
□ Forbidden region



a) FAIL. A neighboring wire is present inside the forbidden region.



b) FAIL. The notch length (0.11) is greater than the minimum of the two spacing values (0.1). Therefore, the constraint applies, but is not met (a neighboring wire is present inside the forbidden region).

Virtuoso Technology Data Constraints Reference

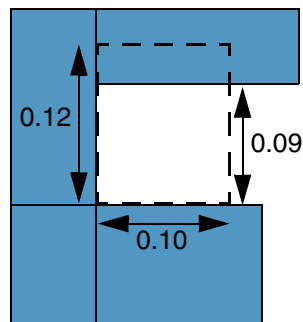
Spacing Constraints (One Layer)

Example 2: *minConcaveCornerSpacing* with *exceptNotchLength*

A neighboring wire must not be present inside a forbidden region that extends 0.1 horizontally and 0.12 vertically from a concave corner. The constraint does not apply to a U-shaped notch if the notch length is less than 0.08.

```
spacingTables (
  ( minConcaveCornerSpacing "Metal1"
    (( "width" nil nil )
      'exceptNotchLength 0.08
    )
    (
      0.0 ((0.1 0.12))
    )
  )
) ;spacingTables
```

■ Metal1
□ Forbidden region



FAIL. The notch length is 0.09 (>0.08). Therefore, the constraint applies, but is not met (a neighboring wire is present inside the forbidden region). The constraint would not apply if 'exceptNotch' was specified.

minConvexCornerSpacing

```
spacings (
    ( minConvexCornerSpacing tx_layer
      'sideExclusionRegion (f_sideExt f_orthogonalExt)
      ['lowCornerSpacing f_lowCornerSpacing]
      ['sameSideExtension f_extension
      | ['edgeForwardExtension f_edgeFwdExt
        'cornerExtensionRegion (f_cornerFwdExt f_cornerBkExt)
        'spanLength f_spanLength
        'minOppositeWidth f_minOppWidth
        'oppositeExtension (f_oppSideExt f_oppFwdExt1 f_oppFwdExt2)
      ]
      | 'middleWires x_middleWires ['includeOverlap
        ['backwardWire (f_sideExt f_backwardExt)]
      ]
      f_spacing
    )
) ;spacings
```

Specifies the minimum orthogonal spacing on a layer between two convex corners of two parallel shapes or between a convex corner and a neighboring shape.

The constraint applies in the following scenarios.

Scenario 1

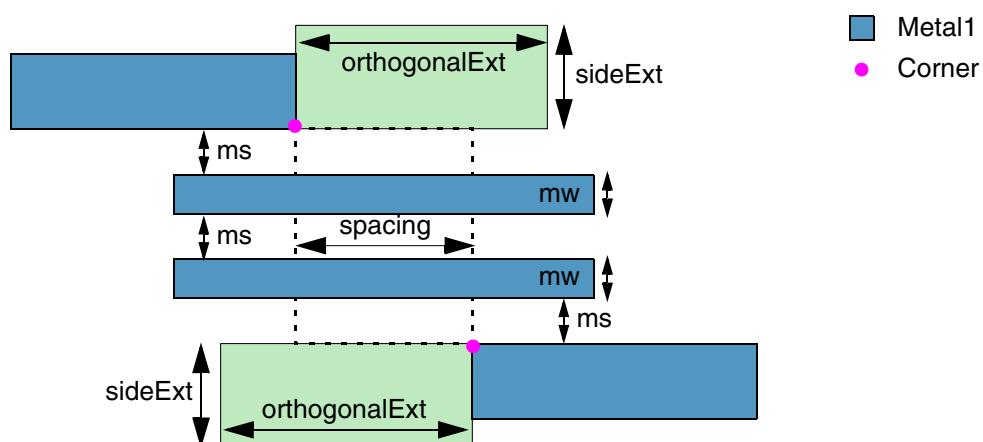
The constraint applies between the convex corners of two parallel shapes facing each other if all of the following conditions are met:

- The parallel shapes have between them two minimum-width (m_w) shapes at minimum spacing (m_s) from each other and from the parallel shapes, running across the region between the two convex corners.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

- No neighboring shapes overlap the two exclusion regions defined by *sideExt* and *orthogonalExt* (green boxes).



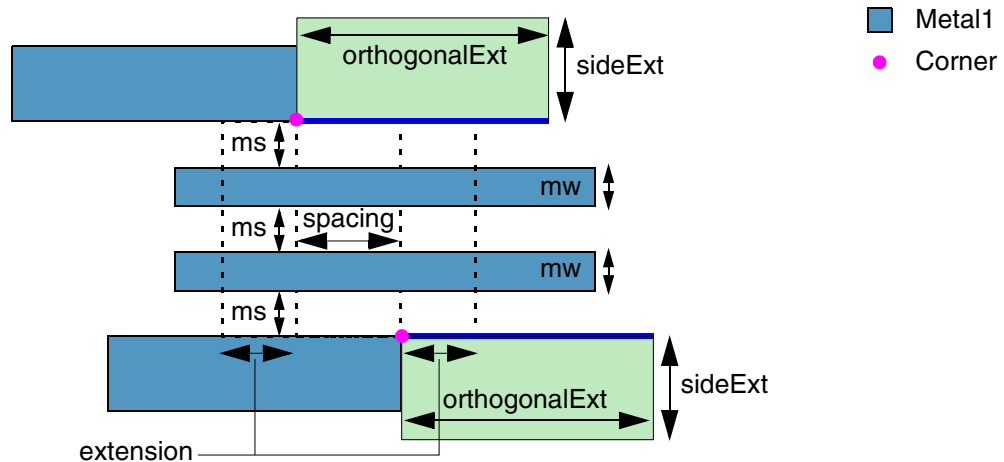
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Scenario 2

The constraint applies between the convex corners of two parallel shapes that do not face each other if the following conditions are met:

- The parallel shapes have between them two minimum-width (mw) shapes at minimum spacing (ms) from each other and from the parallel shapes, running across the region between the two convex corners. The edges of the region are extended by *extension* along the direction of the parallel shapes.
- No neighboring shapes overlap the two exclusion regions defined by *sideExt* and *orthogonalExt* (green boxes).
- No neighboring shapes are coincident with the two blue edges.



Virtuoso Technology Data Constraints Reference

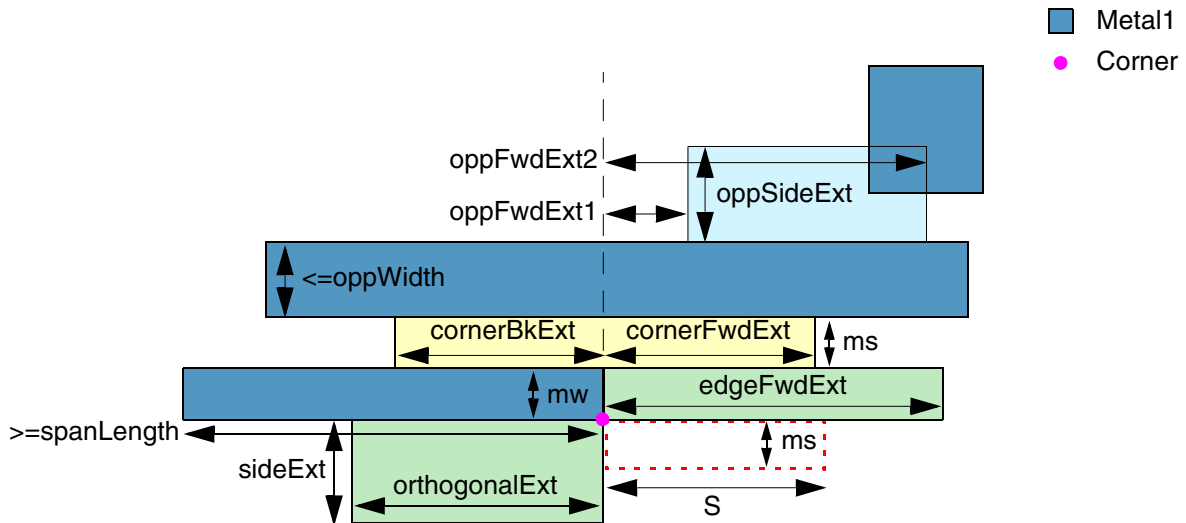
Spacing Constraints (One Layer)

Scenario 3

The constraint applies if all of the following conditions are met:

- mw is the minimum width and ms is the minimum spacing.
- The length of the shape with the convex corner is greater than or equal to $spanLength$.
- No neighboring shapes overlap the two exclusion regions defined by $sideExt$ and $orthogonalExt$ (green boxes).
- A shape with width less than or equal to $oppWidth$ fully abuts the yellow region defined by $cornerBkExt$ and $cornerFwdExt$.
- Another shape overlaps or abuts the light blue region defined by $oppSideExt$, $oppFwdExt1$, and $oppFwdExt2$.

When all conditions listed above are met, a violation occurs if a neighboring shape overlaps the red dotted box.



Virtuoso Technology Data Constraints Reference

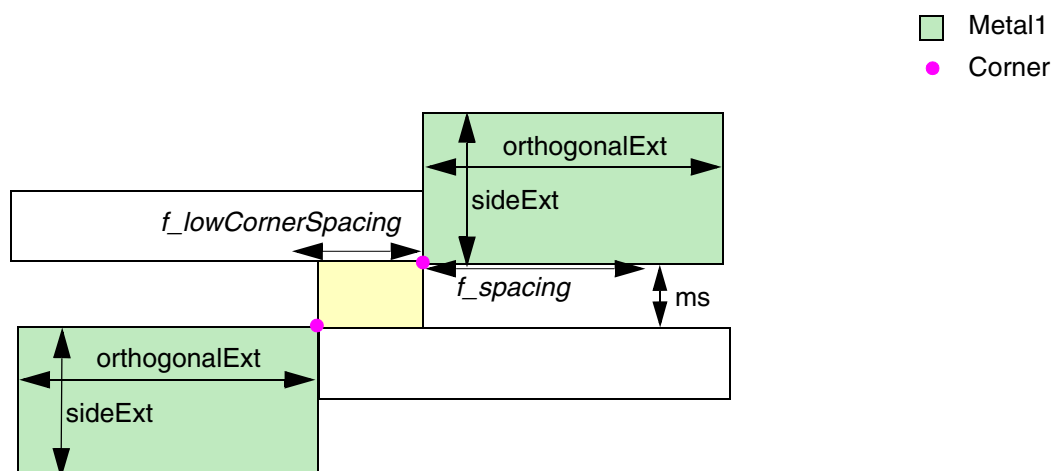
Spacing Constraints (One Layer)

Scenario 4

The constraint applies if all of the following conditions are met:

- ms is the minimum spacing.
- $f\_middleWires$ is 0.
- $f\_lowCornerSpacing$ is less than 0.

A violation occurs when the distance between the two convex corners shown in pink is within the absolute value of $f\_lowCornerSpacing$.



Virtuoso Technology Data Constraints Reference

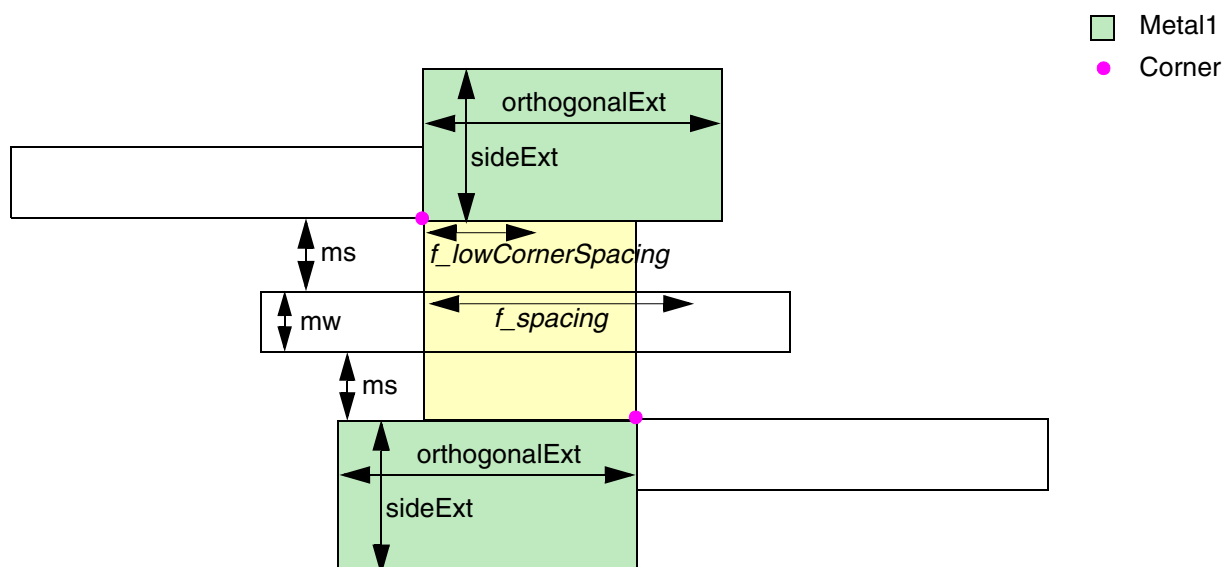
Spacing Constraints (One Layer)

Scenario 5

The constraint applies if all of the following conditions are met:

- ms is the minimum spacing.
- $f\_middleWires$ is 1.
- $f\_lowCornerSpacing$ is greater than 0.

A violation occurs when the distance between the two convex corners shown in pink is less than or equal to $f\_lowCornerSpacing$ or more than or equal to $f\_spacing$.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between convex corners or between a convex corner and a neighboring shape must be greater than or equal to this value.

Parameters

'sideExclusionRegion (*f_sideExt* *f_orthogonalExt*)

The constraint applies only if no neighboring wire overlaps the exclusion region defined by these values. *sideExt* is measured from the convex corner along the shorter side of the shape and *orthogonalExt* is measured perpendicular to the shorter side of the shape.

'lowCornerSpacing *f_lowCornerSpacing*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the distance between two convex corners is less than or equal to *f_lowCornerSpacing* or more than or equal to *f_spacing*. If the specified value is negative, the absolute value is considered.

'sameSideExtension *f_extension*

The region between the two convex corners that do not face each other is extended on both sides by this value.

'edgeForwardExtension *f_edgeFwdExt*

The constraint does not apply if a neighboring shape overlaps the region extending away from the convex corner, with the longer side equal to this value and the shorter side equal to *mw*.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'cornerExtensionRegion (f_cornerFwdExt f_cornerBkExt)`

The constraint applies only if the region defined by these two values is fully abutted by a parallel neighboring shape present along the long side opposite to the convex corner.

cornerFwdExt is measured away from the shape with the convex corner and *cornerBkExt* is measured along the shape with the convex corner.

`'spanLength f_spanLength`

The constraint applies only if the length of the shape with the convex corner is greater than or equal to this value.

`'minOppositeWidth f_minOppWidth`

The constraint applies only if the width of the shape that fully abuts the region defined by `'cornerExtensionRegion` is less than or equal to this value.

`'oppositeExtension (f_oppSideExt f_oppFwdExt1 f_oppFwdExt2)`

The constraint applies only if a shape overlaps the region formed by these three values. *oppSideExt* is measured perpendicular to the shape that fully abuts the region defined by `'cornerExtensionRegion` and *oppFwdExt1* and *oppFwdExt2* are measured away from the short side that forms the convex corner.

`'middleWires x_middleWires`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when *f_extension* is specified and the number of wires between the two corners is this value. The valid values are 0, 1, and 2. When not specified, the default value of 2 is considered.

`'includeOverlap`

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that if neighbor wires merely touches or overlaps with the window formed by two facing convex corners, it is a violation.

`'backwardWire (f_sideExt f_backwardExt)`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Layout Suite EXL and higher tiers) Forms a window between the opposite corners of the two facing convex corners of the constraint by extending the *f_sideExt* along the edge and going backward by the *f_backwardExt* distance in the orthogonal direction.

Examples

Example 1: minConvexCornerSpacing with sideExclusionRegion

Example 2: minConvexCornerSpacing with sideExclusionRegion and sameSideExtension

Example 3: minConvexCornerSpacing with sideExclusionRegion, edgeForwardExtension, cornerExtensionRegion, spanLength, minOppositeWidth, and oppositeExtension

Example 4: minConvexCornerSpacing with sideExclusionRegion, lowCornerSpacing, middleWires, and includeOverlap

Example 1: minConvexCornerSpacing with sideExclusionRegion

```
spacings (
  ( minConvexCornerSpacing "Metall"
    'sideExclusionRegion (0.3 0.8)
    0.6
  )
) ;spacings
```

Example 2: minConvexCornerSpacing with sideExclusionRegion and sameSideExtension

```
spacings (
  ( minConvexCornerSpacing "Metall"
    'sideExclusionRegion (0.3 0.8)
    'sameSideExtension 0.2
    0.6
  )
) ;spacings
```

Example 3: minConvexCornerSpacing with sideExclusionRegion, edgeForwardExtension, cornerExtensionRegion, spanLength, minOppositeWidth, and oppositeExtension

```
spacings (
  ( minConvexCornerSpacing "Metall"
    'sideExclusionRegion (0.3 0.8)
    'edgeForwardExtension 0.7
    'cornerExtensionRegion (0.6 0.6)
    'spanLength 1.3
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
        'minOppWidth 0.1
        'oppExtension (0.2 0.3 0.7)
        0.5
    )
) ;spacings
```

Example 4: minConvexCornerSpacing with sideExclusionRegion, lowCornerSpacing, middleWires , and includeOverlap

```
spacings (
    (minConvexCornerSpacing "Metall"
        'sideExclusionRegion (0.03 0.07)
        'lowCornerSpacing 0.05
        'middleWires 1
        'includeOverlap
        0.08
    )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minCornerSpacing (One layer)

```
spacingTables (
  ( minCornerSpacing tx_layer
    (( "width" nil nil )
      {'convex
        ['endOfLineWidth f_width
          ['exceptJogLength f_jogLength
            ['useEdgeLength] ['treatLAsJog]
            {'horizontalJog | 'verticalJog}
            'jogWidth (f_jogWidth f_wireWidth)
          ]
          ['eolOnly g_eolWidth]
        ]
        ['sameMask]
        ['withinCornerOnly f_within {'horizontal | 'vertical}
          | 'cornerToCorner ['orthogonalPair]]
          | 'concave
        ]
        ['minLength f_minLength]
        ['exceptNotch | 'exceptNotchLength f_notchLength]
      }
      ['exceptSameNet | 'exceptSameMetal]
    )
    (g_table)
  )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) Specifies the spacing between a corner, either concave or convex, and an edge or another corner. The spacing requirement can vary based on the end-of-line width of the shape for which the constraint must be satisfied.

For convex corners, the constraint applies only if the parallel run length between the two neighboring shapes is less than or equal to zero.

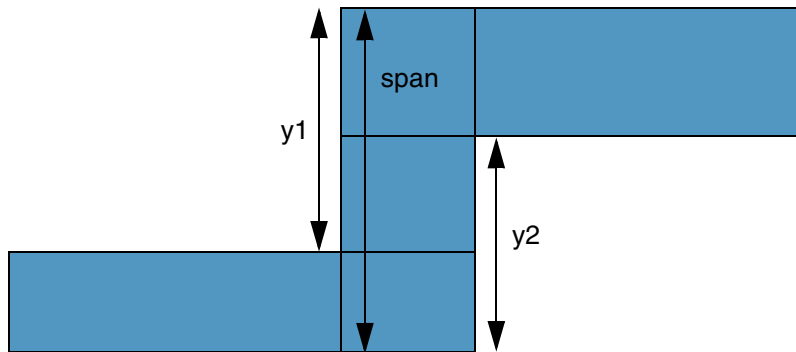
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Optionally, you can use the `'useEdgeLength` parameter to specify how the length specified by `'exceptJogLength` should be measured.

Preferred routing direction: Horizontal

■ Metal2



In the figure above, both $y1$ and $y2$ must be less than *length* for the "Z" shape to qualify for the `'exceptJogLength` exemption. If `'useEdgeLength` is not specified, the span (length of the jog) must be less than *length*.

Values

`tx_layer`

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

`"width" nil nil`

This identifies the index for *table*.

`g_table`

The first value in the table is the width and the second value is the spacing requirement that must be satisfied if the end-of-line width of the shape is greater than the specified value.

Type: A 1-D table specifying floating-point width and spacing values

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'endOfLineWidth f_width`

The constraint does not apply if both shapes have end-of-line widths less than this value.

`'convex | 'concave`

The corner type.

Type: Boolean

`'exceptJogLength f_jogLength`

The constraint does not apply between a Z-shaped jog and a neighboring wire with end-of-line width greater than or equal to *wireWidth* and direction perpendicular to the direction in which the length of the jog is measured if the following conditions are satisfied:

- The length of the jog is less than this value.
- The width of the jog is equal to *jogWidth*.
- The width of the adjacent wires is greater than or equal to *wireWidth* and their projected parallel run length in the direction of the jog is less than zero.

`'useEdgeLength`

(Virtuoso Advanced Node for Layout Only) The length of the edges in the wrong-way direction (non-preferred routing direction) of a wrong-way jog in a Z shape must be less than *length*. Otherwise, the span (length of the jog) must be less than *length*.

Type: Boolean

`'treatLAsJog`

The jog exemption for a Z-shaped jog also applies to an L-shaped jog.

Type: Boolean

`'horizontalJog | 'verticalJog`

The direction in which the length of the jog is measured.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'jogWidth f_jogWidth f_wireWidth`

The constraint does not apply to a Z- or L-shaped jog with length less than *jogLength* in the specified direction if the following conditions are satisfied:

- The width of the jog is equal to *jogWidth*.
- The end-of-line width of the adjacent wires and the neighboring wire is greater than or equal to *wireWidth*.

`'eolOnly`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only between two end-of-line edges with width less than the specified value.

`'sameMask`

(Virtuoso Advanced Node for Layout Only) The constraint applies only to shapes on the same mask.

Type: Boolean

`'withinCornerOnly f_within`

The constraint applies only to neighboring corners. These corners must not be present inside a search window whose one dimension is equal to the constraint spacing in the specified direction and the other dimension is equal to this value.

`'horizontal | 'vertical`

The direction in which the constraint spacing is applied while defining the search window inside which neighboring corners are not allowed.

Type: Boolean

`'cornerToCorner`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only between two facing convex corners.

`'orthogonalPair`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only on corners of two orthogonal direction wires.

`'minLength f_minLength`

The constraint applies only if the edges of a concave corner are greater than or equal to this value in length.

`'exceptNotch`

The constraint does not apply to a notch (U shape).

Type: Boolean

`'exceptNotchLength f_notchLength`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Advanced Node for Layout Only) The constraint does not apply to a notch if the notch length is less than this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'exceptSameNet | 'exceptSameMetal

The constraint does not apply to certain connectivity types.

- 'exceptSameNet: The constraint does not apply to same-net shapes.
- 'exceptSameMetal: The constraint does not apply to contiguous same-metal shapes.

Type: Boolean

Examples

- Example 1: minCornerSpacing with convex, exceptEndOfLineWidth, concave, and exceptNotch
- Example 2: minCornerSpacing with convex, exceptEndOfLineWidth, exceptJogLength, concave, and minLength
- Example 3: minCornerSpacing with convex, exceptEndOfLineWidth, exceptJogLength, and treatAsJog
- Example 4: minCornerSpacing with convex and withinCornerOnly

Virtuoso Technology Data Constraints Reference

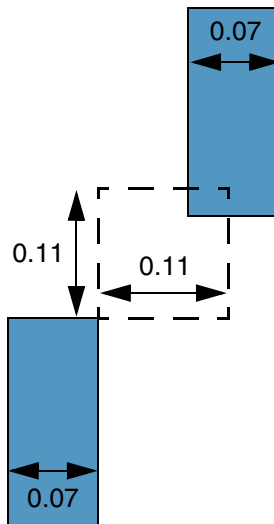
Spacing Constraints (One Layer)

Example 1: *minCornerSpacing* with *convex*, *exceptEndOfLineWidth*, *concave*, and *exceptNotch*

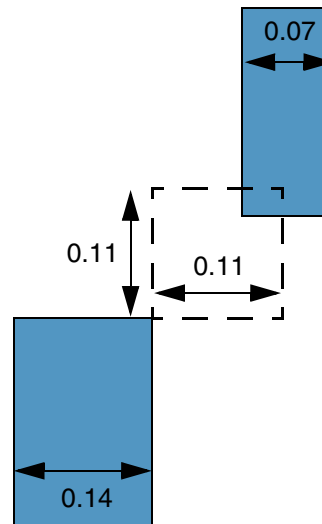
The minimum spacing requirement for a convex corner is 0.11, except when the two shapes have end-of-line widths less than 0.09 or the parallel run length between them is less than or equal to zero. The minimum spacing requirement for a concave corner is 0.15, except when a concave edge is less than 0.05 in length or the shape is a notch.

```
spacingTables (  
  ( minCornerSpacing "Metal1"  
    (( "width" nil nil )  
      'convex  
      'exceptEndOfLineWidth 0.09  
    )  
    ( 0.00 0.11  
    )  
  )  
  ( minCornerSpacing "Metal1"  
    (( "width" nil nil )  
      'concave  
      'exceptNotch  
    )  
    ( 0.00 0.15  
    )  
  )  
) ;spacingTables
```

■ Metal1



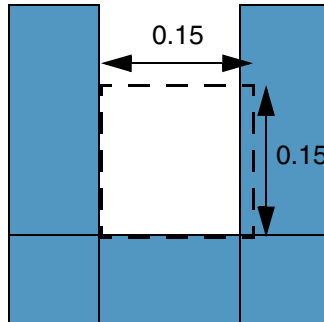
a) The constraint does not apply because the end-of-line width of both shapes is 0.07 (<0.09).



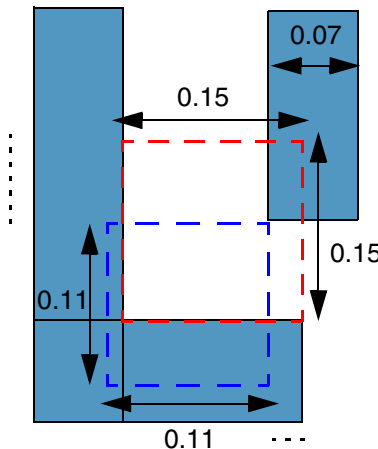
b) FAIL. The end-of-line width of one of the shapes is 0.14 (>0.09). Therefore, the spacing between the two shapes must be at least 0.11.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply. A violation would occur if 'exceptNotch' was not specified.



d) FAIL. The convex corner minimum spacing (blue dotted line) does not apply because the parallel run length between the convex corner and the neighboring shapes, both vertical and horizontal (black dotted lines), is greater than zero. However, the concave corner spacing rule does apply and requires a minimum spacing of 0.15 (red dotted line).

Example 2: minCornerSpacing with convex, exceptEndOfLineWidth, exceptJogLength, concave, and minLength

The minimum spacing requirement for a convex corner is 0.11, except when the two shapes have end-of-line widths less than 0.09 or the parallel run length between them is less than or equal to zero. A Z-shaped jog is exempt from the spacing requirement if its length is less than 0.17 in the horizontal direction, width is 0.09, and the width of the adjacent wires and the neighboring wire perpendicular to the jog direction is greater than or equal to 0.05.

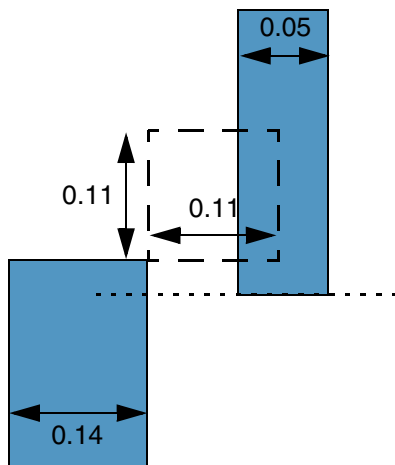
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

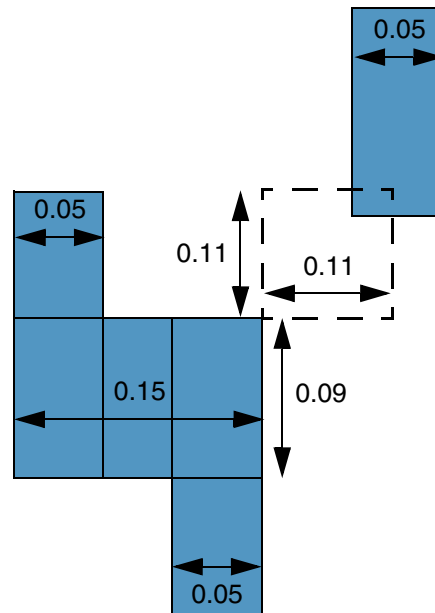
The minimum spacing requirement for a concave corner is 0.15, except when a concave edge is less than 0.05 in length.

```
spacingTables (
  ( minCornerSpacing "Metal1"
    (( "width" nil nil )
      'convex
      'exceptEndOfLineWidth 0.09
      'exceptJogLength 0.17
      'horizontalJog
      'jogWidth 0.09 0.05
    )
    (
      0.00 0.11
    )
  )
  ( minCornerSpacing "Metal1"
    (( "width" nil nil )
      'concave 'minLength 0.05
    )
    (
      0.00 0.15
    )
  )
) ;spacingTables
```

■ Metal1



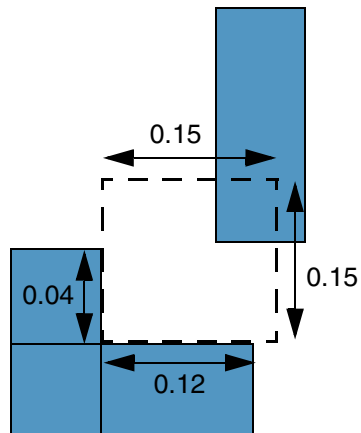
a) The constraint does not apply. The parallel run length between the convex corner and the shape on the right is greater than zero.



b) The constraint does not apply because the length of the Z-shaped jog in the horizontal direction is 0.15 (< 0.17), the width is 0.09, and the width of the adjacent wires and the neighboring wire is 0.05.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply because the vertical concave edge is 0.04 long (< 0.05).

Virtuoso Technology Data Constraints Reference

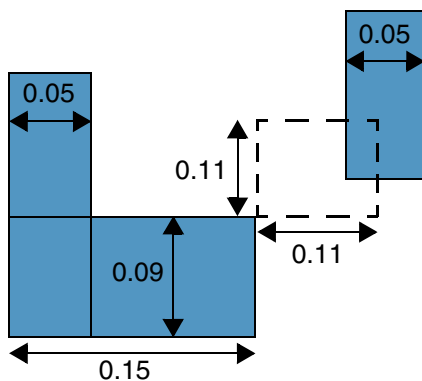
Spacing Constraints (One Layer)

Example 3: minCornerSpacing with convex, exceptEndOfLineWidth, exceptJogLength, and treatAsJog

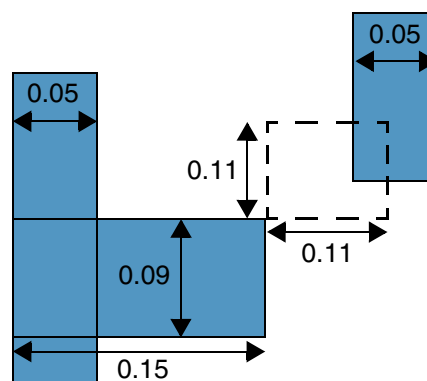
The minimum spacing requirement for a convex corner is 0.11, except when the two shapes have end-of-line widths less than 0.09 or the parallel run length between them is less than or equal to zero. A Z- or L-shaped jog is exempt from the spacing requirement if its length is less than 0.17 in the horizontal direction, width is 0.09, and the width of the adjacent wires and a neighboring wire perpendicular to the jog direction is greater than or equal to 0.05.

```
spacingTables (
  ( minCornerSpacing "Metal1"
    (( "width" nil nil )
      'convex
      'exceptEndOfLineWidth 0.09
      'exceptJogLength 0.17 'treatLAsJog
      'verticalJog
      'jogWidth 0.09 0.05
    )
    (
      0.00 0.11
    )
  )
) ;spacingTables
```

■ Metal1



a) The constraint does not apply because the length of the L-shaped jog in the horizontal direction is 0.15 (< 0.17), the width is 0.09, and the width of the adjacent wires and the neighboring wire is 0.05.



b) FAIL. The spacing between the T-shaped jog and the neighboring wire must be at least 0.11. (Only Z- and L-shaped jogs are exempt from the spacing requirement.)

Virtuoso Technology Data Constraints Reference

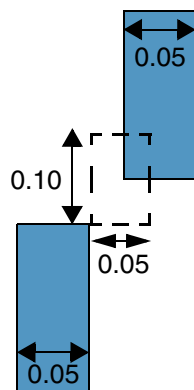
Spacing Constraints (One Layer)

Example 4: *minCornerSpacing* with *convex* and *withinCornerOnly*

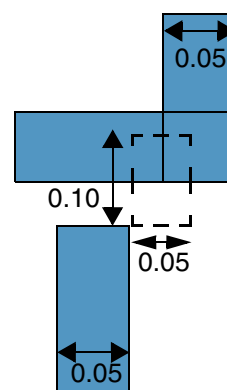
The minimum spacing between a convex corner and a neighboring corner is 0.10 in the vertical direction and 0.05 in the horizontal direction.

```
spacingTables (  
  ( minCornerSpacing "Metal1"  
    (( "width" nil nil )  
      'convex  
      'withinCornerOnly 0.05 'vertical  
    )  
    (  
      0.0 0.10  
    )  
  )  
) ;spacingTables
```

■ Metal1



a) FAIL. A neighboring corner is present inside the search window.

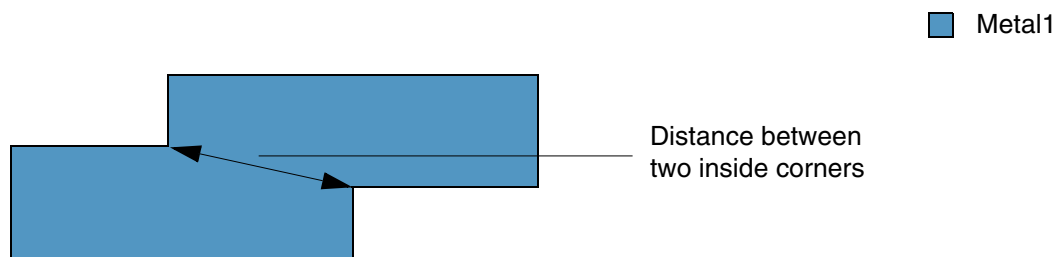


b) The constraint does not apply. No corner is found inside the search window.

minCornerToCornerDistance

```
spacings (
    ( minCornerToCornerDistance tx_layer
      ['extensionLength f_extensionLength ['oppositeDirection]]
      ['exceptJog']
      f_distance
    )
) ;spacings
```

Specifies the minimum distance between the inside corners of a shape.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_distance</i>	The distance between the inside corners of a shape must be greater than or equal to this value.

Parameters

<i>'extensionLength f_extensionLength</i>	(Virtuoso Layout Suite EXL and higher tiers) Specifies the length of the extension that needs to be created from the internal concave corners. Distance checking would take place between the parallel extensions of the two corners where the extensions have parallel run length more than or equal to zero.
<i>'oppositeDirection</i>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when the extensions are in opposite directions. Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'exceptJog

(Virtuoso Layout Suite EXL and higher tiers) Specifies that the rule does not apply to jogs.

Virtuoso Technology Data Constraints Reference

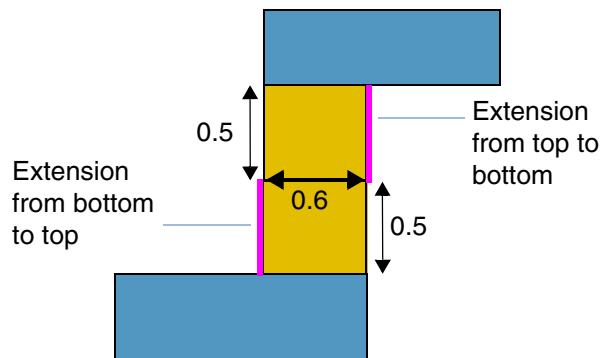
Spacing Constraints (One Layer)

Example

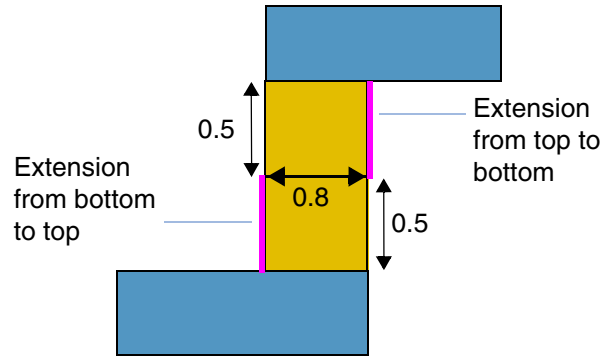
The distance between extensions of a shape must be greater than or equal to 0.7 when extensions of 0.5 from the corners face each other in opposite directions.

```
spacings(  
  ( minCornerToCornerDistance "Metal1"  
    'extensionLength 0.5  
    'oppositeDirection  
    0.7  
  )  
) ;spacings
```

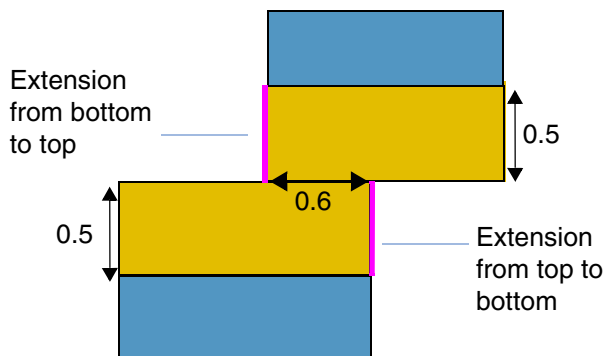
■ Metal1



FAIL. The extensions of 0.5 from the two concave corners are facing each other in opposite directions and overlapping each other. However, the distance between the extensions is 0.6, which is less than 0.7.



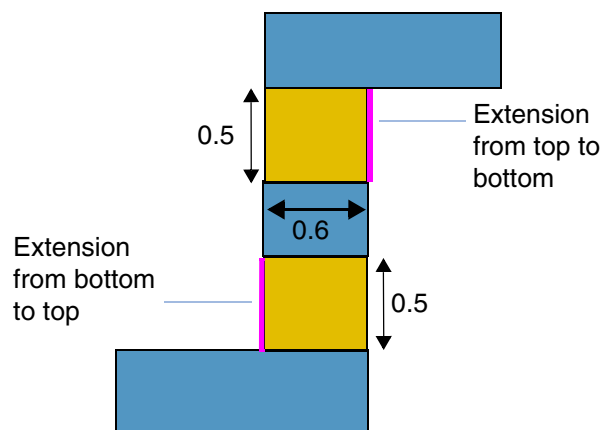
PASS. The extensions of 0.5 from the two concave corners are facing each other in opposite directions and overlapping each other. The distance between the extensions is 0.8, which is more than 0.7.



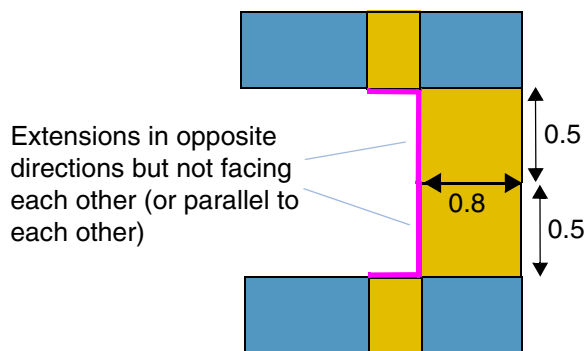
FAIL. The extensions of 0.5 from the two concave corners are facing each other in opposite directions and PRL is more than or equal to 0. However, the distance between the extensions is 0.6.

Virtuoso Technology Data Constraints Reference

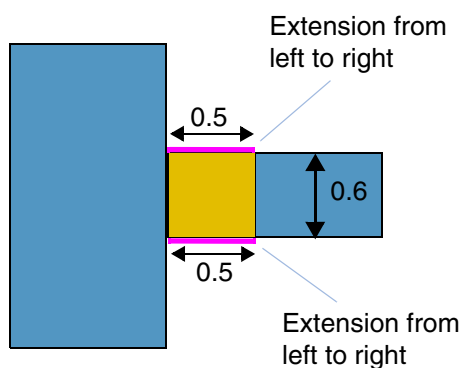
Spacing Constraints (One Layer)



PASS. The constraint does not apply because the extensions of 0.5 from the two concave corners do not have PRL more than or equal to 0.



PASS. The constraint does not apply because the extensions of 0.5 from the two concave corners are not facing (or parallel) to each other.



PASS. The constraint does not apply because the extensions of 0.5 from the two concave corners are not in opposite directions.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minCornerVoltageSpacing

```
spacings (
    ( minCornerVoltageSpacing tx_layer
      'prlRange f_prlRange
      'deltaVoltage f_voltage
      ['subtractTrim]
      ['endOfLineWidth f_eolWidth]
      f_minSpacing
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the minimum spacing required based on the voltage between the corners of shapes on a specified layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_minSpacing</i>	The minimum spacing required.

Parameters

<i>'prlRange f_prlRange</i>	The constraint applies only if the parallel run length between the two shapes falls in this range. Type: Floating-point values specifying a <u>range</u> of parallel run lengths that trigger the constraint

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'deltaVoltage *f_voltage*

The constraint applies only if the maximum voltage difference between the two shapes is greater than this value.

'subtractTrim

If true, and the layer is trimmable, the trim will be subtracted from the layer before checking is performed.

Type: Boolean

'endOfLineWidth *f_eolWidth*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to end-of-line edges with width less than or equal to this value.

Examples

```
spacings (
  ( minCornerVoltageSpacing "Metall"
    'prlRange "[-0.06 0]"
    'subtractTrim
    'deltaVoltage 0.12 0.22
  )
) ;spacings
```

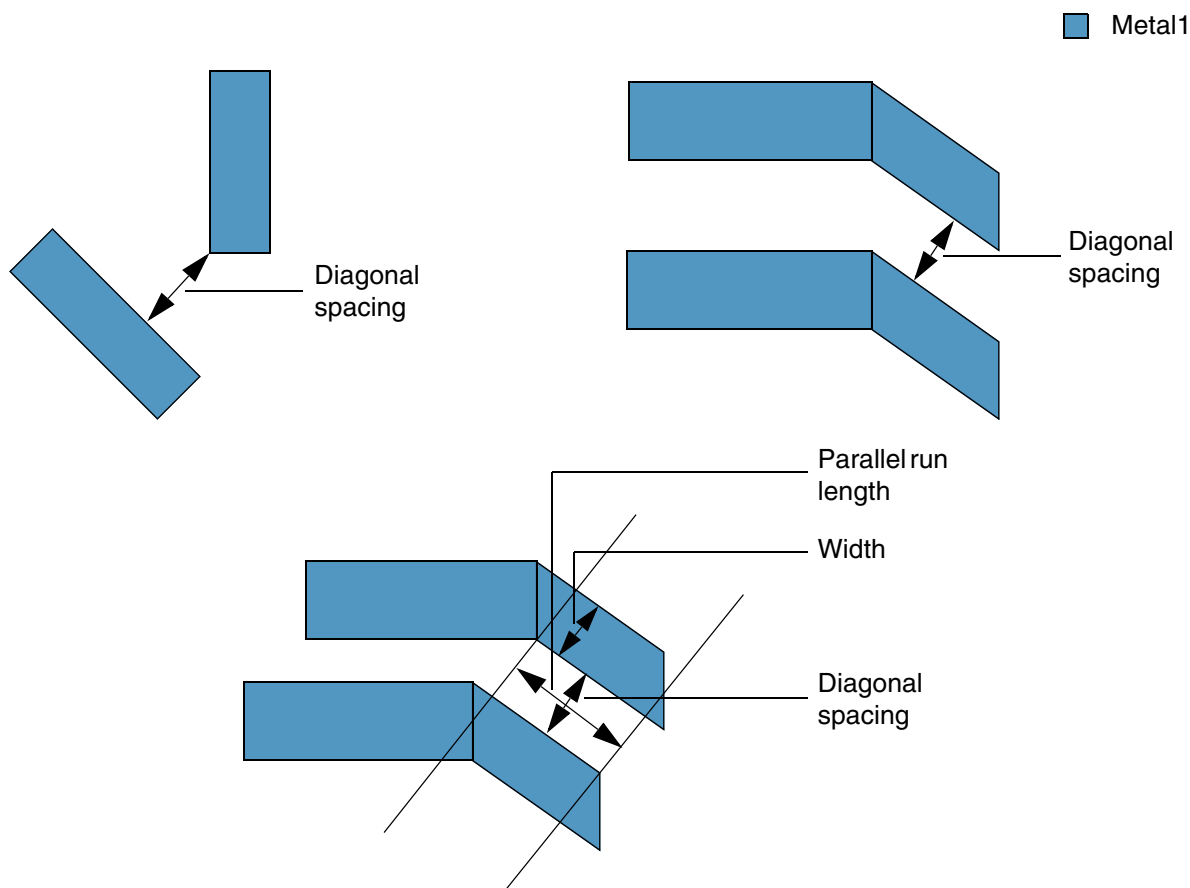
minDiagonalSpacing

```

spacings (
    ( minDiagonalSpacing tx_layer
      f_spacing
    )
) ;spacings

spacingTables (
    ( minDiagonalSpacing tx_layer
      (( "width" nil nil ["length" nil nil] )
        [f_default]
      )
      (g_table)
    )
) ;spacingTables
    
```

Specifies the minimum diagonal spacing between two shapes on a layer. The distance must be calculated as the Euclidean diagonal distance, and not as the Manhattan distance.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum required diagonal spacing.
"width" nil nil ["length" nil nil]	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of a 1-D table is as follows: <i>(f_index f_value)</i> where, <i>f_index</i> is the width of the wider of the two shapes and <i>f_value</i> is the minimum diagonal spacing required between two shapes when the width is greater than or equal to the corresponding index. The format of a 2-D table is as follows: <i>((f_index1 f_index2) f_value)</i> where, <i>f_index1</i> is the width of the wider of the two shapes and <i>f_index2</i> is the parallel run length between the two shapes. <i>f_value</i> is the minimum diagonal spacing required between two shapes when both width and parallel run length are greater than or equal to the corresponding indexes. Note: The parallel run length between the two shapes is calculated as the lengthwise distance for which the two shapes are directly orthogonal (that is, a line running perpendicular to the length must intersect both shapes). Type: A 1-D table specifying floating-point width and diagonal spacing values, or a 2-D table specifying floating-point width, parallel run length, and diagonal spacing values

Parameters

<i>f_default</i>	The spacing value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The following diagonal spacing requirement between two shapes on a layer must be satisfied:

- On layer Poly1, the diagonal spacing must be at least 0.7.
- On layer Metal2, the diagonal spacing must be at least equal to the value of the technology parameter `mindiaspace1`.

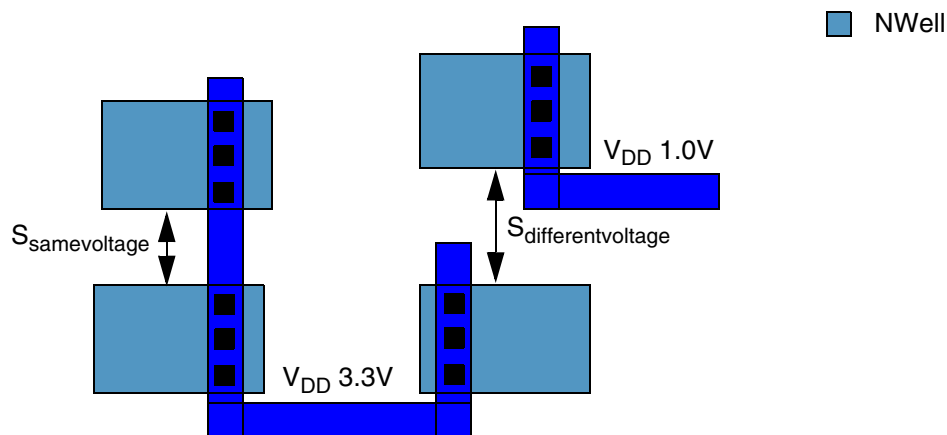
```
spacings(  
  ( minDiagonalSpacing "Poly1"  
    0.7  
  )  
  ( minDiagonalSpacing "Metal2"  
    techParam("mindiaspace1")  
  )  
) ;spacings
```


minDiffNetSpacing

```
spacings (
    ( minDiffNetSpacing tx_layer
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between shapes on nets at different voltage levels on the specified layer.

This constraint applies to wells, transistors, and devices. When two geometries, typically two "NWell" shapes, are at different voltages, a larger spacing is required between the two shapes. The larger spacing prevents potential leakage problems through the substrate.



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_spacing

The spacing between two shapes that are at different voltage levels must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The following spacing requirement must be satisfied if a voltage difference exists between two nets on a layer:

- The spacing between two Metal1 shapes must be at least 1.0.
- The spacing between two Metal2 shapes must be at least equal to the value of the technology parameter `mindiffnetspace1`.

```
spacings (
  ( minDiffNetSpacing "Metal1"
    1.0
  )
  ( minDiffNetSpacing "Metal2"
    techParam("mindiffnetspace1")
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minEdgeLengthSpacing

```
spacings (
    ( minEdgeLengthSpacing tx_layer
      'length f_length ['prl f_parallelRunLength]
      ['horizontalEdge | 'verticalEdge]
      f_spacing
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between a shape on the specified layer and the short edge of a second shape on the same layer. The constraint applies only if the length of the short edge is less than the specified value.

Optionally, the constraint can be applied to edges in the specified direction and to edges that have parallel run length greater than or equal to the specified value.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum required spacing between shapes.

Parameters

<i>'length f_length</i>	The constraint applies only to the edges with length less than this value.
<i>'prl f_parallelRunLength</i>	The constraint applies only if the parallel run length between the two edges is greater than or equal to this value.
<i>'horizontalEdge 'verticalEdge</i>	The constraint applies only to the edges in the specified direction. The default is any direction. Type: Boolean

Virtuoso Technology Data Constraints Reference

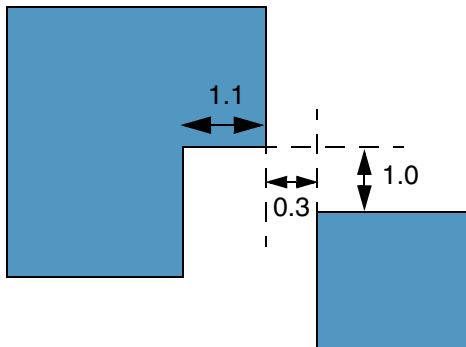
Spacing Constraints (One Layer)

Example

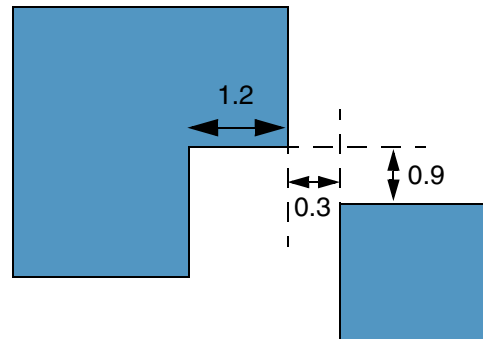
The spacing between a shape and the horizontal short edge of another shape on Metal1 must be at least 1.0 if the short edge is less than 1.2 long and the parallel run length between the two edges is greater than or equal to -0.3.

```
spacings (  
  ( minEdgeLengthSpacing "Metal1"  
    'length 1.2 'prl -0.03  
    'horizontalEdge  
    1.0  
  )  
) ;spacings
```

■ Metal1



a) PASS. The length of the horizontal short edge of the polygon is 1.1 (< 1.2) and the parallel run length between the two edges is -0.3. The spacing between the two edges is 1.0.



b) The constraint does not apply because the length of the short edge of the polygon is 1.2. For the constraint to apply, it must be less than 1.2.

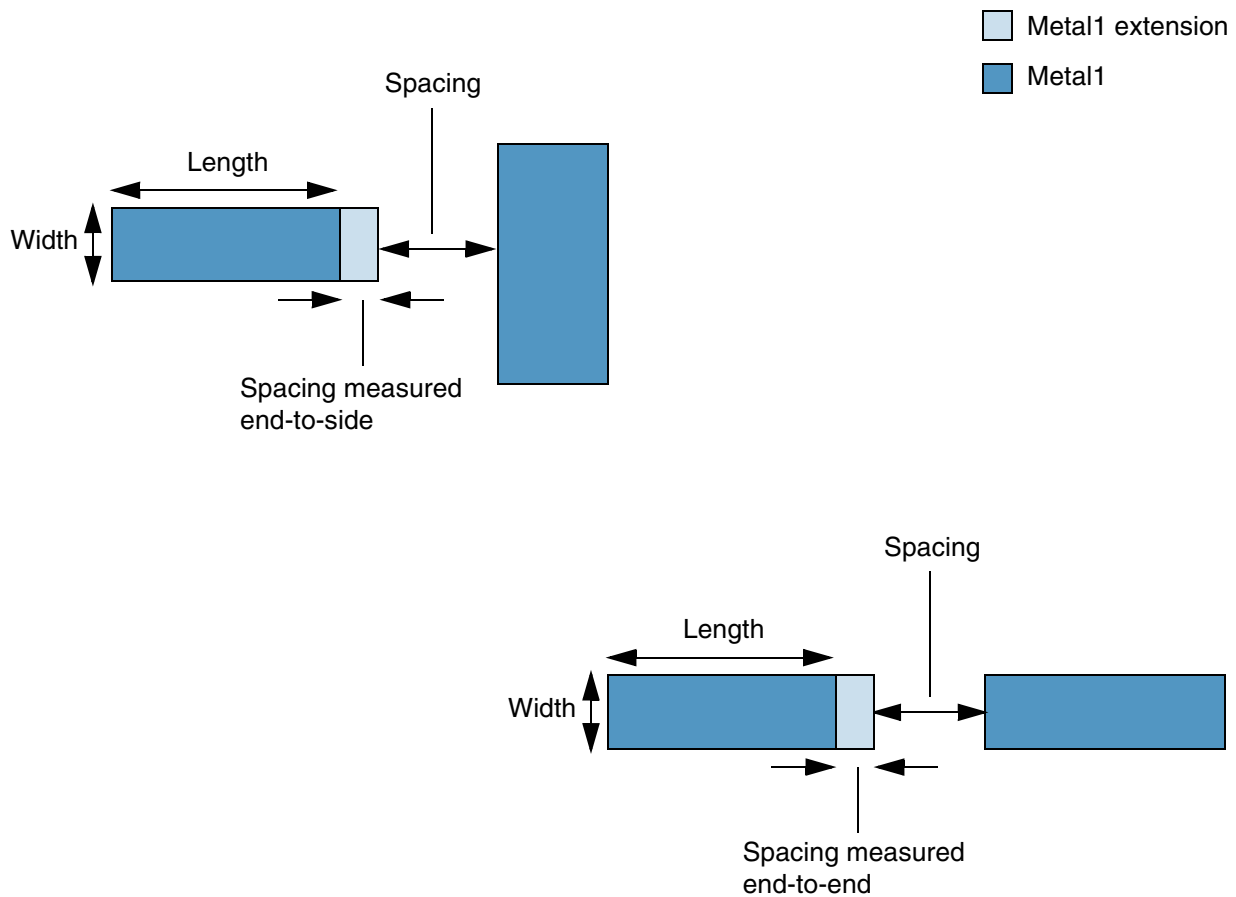
minEndOfLineExtensionSpacing

```

spacings (
    ( minEndOfLineExtensionSpacing tx_layer
      ['length f_length ['twoSides]]
      ['sameMask]
      ['negativePRL ['nonEol]]
      ['otherWidth f_otherWidth]
      'extensions ((( "width" )) (g_table))
      f_spacing
    )
) ;spacings

```

Specifies the minimum spacing between an end-of-line edge and its neighboring edges. The spacing is measured after applying the required extension to the end-of-line edge, as shown below.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing requirement that must be satisfied.

Parameters

<code>'length</code> <i>f_length</i>	The constraint applies only if the length is greater than or equal to this value.
<code>'twosides</code>	The constraint applies only if the length of both sides is greater than or equal to <i>length</i> . Type: Boolean
<code>'sameMask</code>	(Virtuoso Advanced Node for Layout Standard) The constraint applies only to same-mask shapes. Type: Boolean
<code>'negativePRL</code>	(Virtuoso Advanced Node for Layout Standard) The constraint applies only if there exists a neighboring edge that has a parallel run length less than zero with the end-of-line edge. Type: Boolean
<code>'nonEol</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies if the neighboring edge that has a parallel run length less than zero with the end-of-line edge is not an end-of-line edge. Type: Boolean
<code>'otherWidth</code> <i>f_otherWidth</i>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the width of the neighboring shape is less than this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
'extensions ((( "width" )) (g_table))
```

Extension value to be applied before spacing is checked.

The *g_table* entries are defined using one of the following formats:

- `(f_width f_extension)`

where, *f_width* is the width of the end-of-line edge and *f_extension* is the extension to be applied (if width is less than *f_width*) before checking for edge-to-edge spacing to any neighboring wires.

- `(f_width (f_extension
[f_endToEndExtension]))`

where, *f_extension* is the extension to be applied if spacing is measured end-to-side, and *f_endToEndExtension* is the extension to be applied if spacing is measured end-to-end.

"width" identifies the index for *table*.

Type: A 1-D table specifying floating-point width and extension values

Examples

- Example 1: minEndOfLineExtensionSpacing
- Example 2: minEndOfLineExtensionSpacing with negativePRL
- Example 3: minEndOfLineExtensionSpacing with nonEol
- Example 4: minEndOfLineExtensionSpacing with otherWidth

Virtuoso Technology Data Constraints Reference

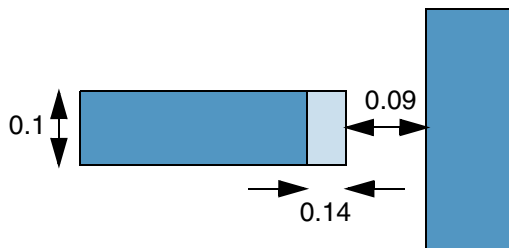
Spacing Constraints (One Layer)

Example 1: minEndOfLineExtensionSpacing

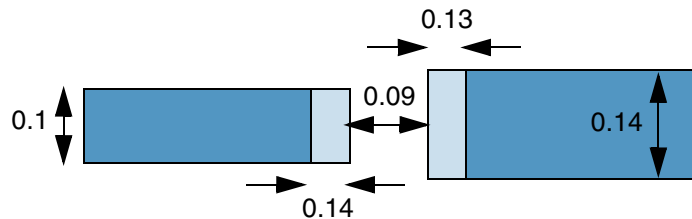
An end-of-line edge with width less than 0.11 requires 0.14 extension, and an end-of-line edge with width greater than or equal to 0.11 and less than 0.15 requires 0.12 extension when spacing is measured end-to-side and 0.13 extension when spacing is measured end-to-end. The edge-to-edge spacing must be at least 0.1.

```
spacings (
  ( minEndOfLineExtensionSpacing "Metal1"
    'extensions ((( "width" )) (0.11 0.14))
    0.1
  )
  ( minEndOfLineExtensionSpacing "Metal1"
    'extensions ((( "width" )) (0.15 (0.12 0.13))
    0.1
  )
) ;spacings
```

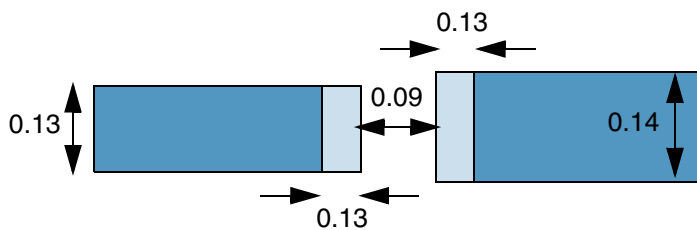
□ Metal1 extension
■ Metal1



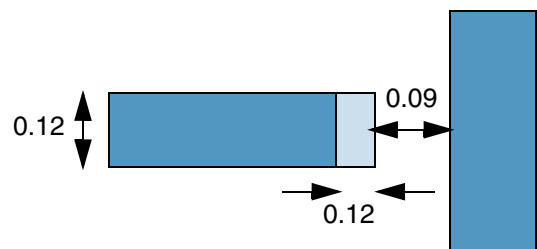
a) FAIL. The spacing measured after applying 0.14 extension is 0.09 (<0.1).



b) FAIL. The spacing measured after applying the required extensions is 0.09 (<0.1). (The left end-of-line width is 0.1 (<0.11), so extension applied is 0.14. The right end-of-line width is 0.14 (>=0.11 and <0.15) and spacing is measured end-to-end, so extension applied is 0.13.)



c) FAIL. The spacing measured after applying the required extensions is 0.09 (<0.1). (Both end-of-line widths are >=0.11 and <0.15 and spacing is measured end-to-end, so extension applied is 0.13.)



d) FAIL. The spacing measured after applying the required extensions is 0.09 (<0.1). (The left end-of-line width is 0.12 (>=0.11 and <0.15) and spacing is measured end-to-side, so extension applied is 0.12.)

Virtuoso Technology Data Constraints Reference

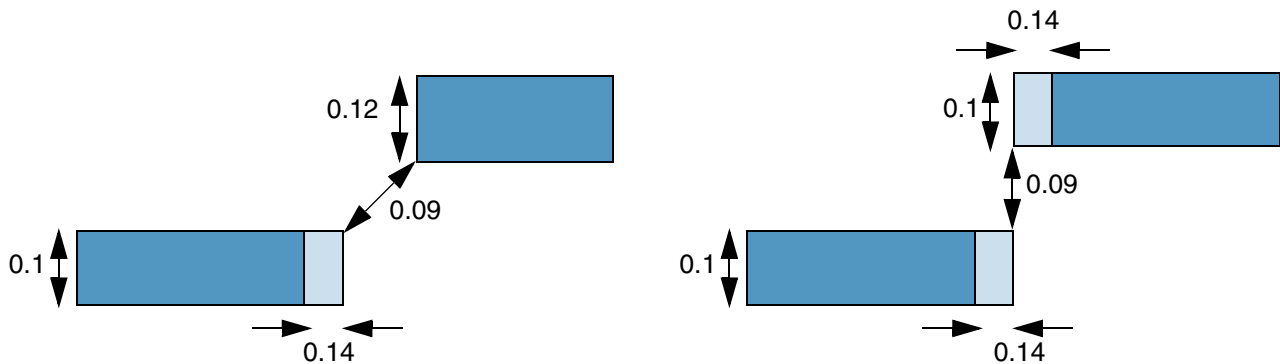
Spacing Constraints (One Layer)

Example 2: *minEndOfLineExtensionSpacing* with *negativePRL*

The minimum spacing between an end-of-line edge and the neighboring edges with which it has a negative parallel run length must be 0.1. If the end-of-line edge is less than 0.11 wide, a 0.14 extension is applied before spacing is checked.

```
spacings (  
  ( minEndOfLineExtensionSpacing "Metal1"  
    'negativePRL  
    'extensions ((( "width" )) (0.11 .14))  
    0.1  
  )  
) ;spacings
```

□ Metal1 extension
■ Metal1



a) FAIL. The end-of-line edge has a parallel run length less than zero with the neighboring edge, but the spacing between them after applying 0.14 extension to the end-of-line edge is 0.09 (< 0.1).

b) The constraint does not apply. The parallel run length between the two edges is zero. The constraint would fail if 'negativePRL' was not specified.

Virtuoso Technology Data Constraints Reference

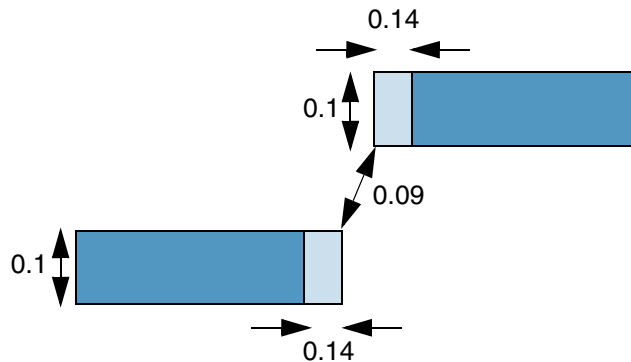
Spacing Constraints (One Layer)

Example 3: *minEndOfLineExtensionSpacing* with *nonEol*

The minimum spacing between an end-of-line edge and a neighboring edge with which it has a negative parallel run length must be 0.1 if the neighboring edge is not an end-of-line edge. A 0.14 extension is applied before the spacing is checked if the end-of-line edge is less than 0.11 wide.

```
spacings (  
  ( minEndOfLineExtensionSpacing "Metal1"  
    'negativePRL 'nonEol  
    'extensions ((("width")) (0.11 .14))  
    0.1  
  )  
) ;spacings
```

□ Metal1 extension
■ Metal1



The constraint does not apply. The neighboring edge that has a parallel run length less than zero with the end-of-line edge is also an end-of-line edge. The constraint would fail if 'nonEol' was not specified.

Virtuoso Technology Data Constraints Reference

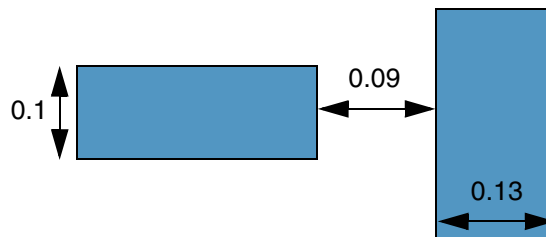
Spacing Constraints (One Layer)

Example 4: minEndOfLineExtensionSpacing with otherWidth

An end-of-line Metal1 edge with width less than 0.11 requires zero extension, and the edge-to-edge spacing between two neighboring Metal1 shapes must be at least 0.1, provided the width of the neighboring shape is less than 0.12.

```
spacings (  
  ( minEndOfLineExtensionSpacing "Metal1"  
    'otherWidth 0.12  
    'extensions (((("width")) (0.11 (0.0 0.0)))  
    0.1  
  )  
) ;spacings
```

■ Metal1



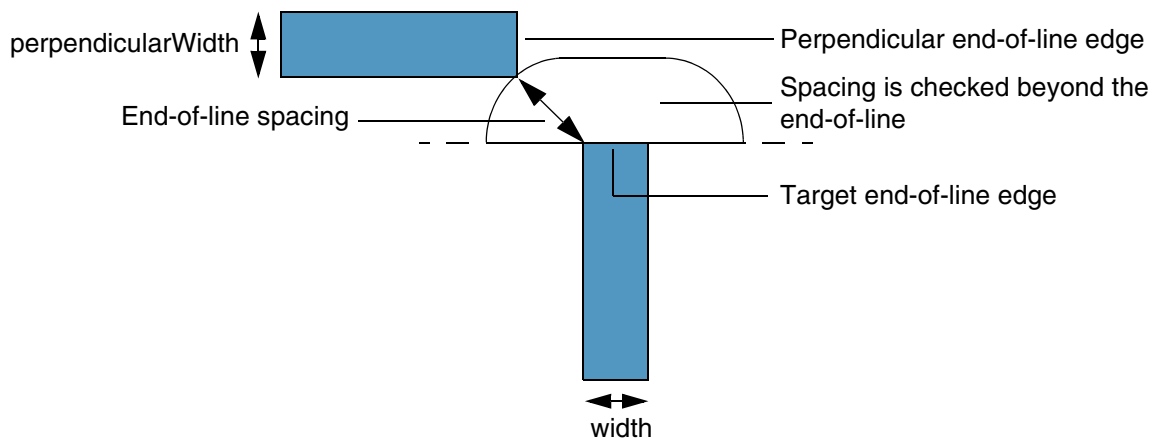
The constraint does not apply because the width of the neighboring wire is 0.13 (>0.12). A violation would occur if 'otherWidth was not specified.

minEndOfLinePerpSpacing

```
spacings (
    ( minEndOfLinePerpSpacing tx_layer
      'width f_width
      'perpWidth f_perpendicularWidth
      ['negativePRL]
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between an end-of-line edge and another edge that lies beyond the end-of-line edge on the same layer and is perpendicular to the end-of-line edge.

■ Metal1



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum required spacing between the two edges.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'width f_width`

The constraint applies only if the width of the end-of-line edge is less than this value.

`'perpWidth f_perpendicularWidth`

The constraint applies only if the width of the perpendicular edge is less than this value.

`'negativePRL`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the perpendicular edge has a parallel run length of less than zero with the end-of-line edge.

Examples

- Example 1: minEndOfLinePerpSpacing with perpWidth
- Example 2: minEndOfLinePerpSpacing with negativePRL

Virtuoso Technology Data Constraints Reference

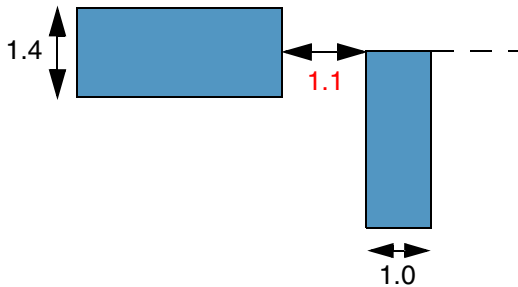
Spacing Constraints (One Layer)

Example 1: *minEndOfLinePerpSpacing* with *perpWidth*

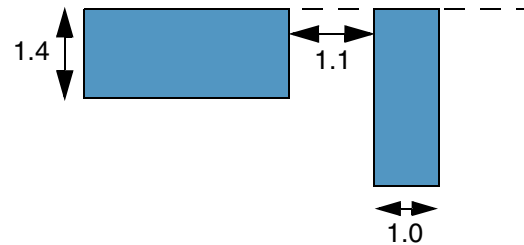
The spacing between an end-of-line edge and a perpendicular edge that lies beyond the end-of-line edge on layer Metal1 must be at least 1.2 if the width of the end-of-line edge is less than 1.1 and the width of the perpendicular edge is less than 1.5.

```
spacings (  
  ( minEndOfLinePerpSpacing "Metal1"  
    'width 1.1  
    'perpWidth 1.5  
    1.2  
  )  
) ;spacings
```

■ Metal1



a) FAIL. The width of the target end-of-line edge is 1.0 (<1.1) and the width of the perpendicular edge is 1.4 (<1.5), but the spacing between the two edges is 1.1 (<1.2).



b) The constraint does not apply because the perpendicular edge is not "seen" by the target end-of-line edge.

Virtuoso Technology Data Constraints Reference

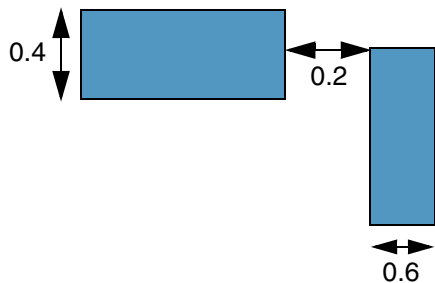
Spacing Constraints (One Layer)

Example 2: *minEndOfLinePerpSpacing* with *negativePRL*

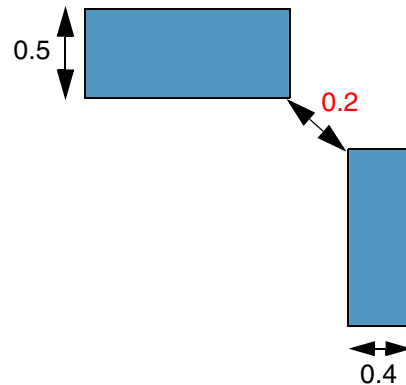
The spacing between an end-of-line edge and a perpendicular edge that lies beyond the end-of-line edge on layer Metal1 must be at least 0.3 if the width of the end-of-line edge is less than 0.5 and the width of the perpendicular edge is less than 0.7. Additionally, the perpendicular edge must have a parallel run length of less than zero with the end-of-line edge.

```
spacings (
  ( minEndOfLinePerpSpacing "Metal1"
    'width 0.5
    'perpWidth 0.7
    'negativePRL
    0.3
  )
) ;spacings
```

■ Metal1



The constraint does not apply because the perpendicular edge has positive parallel run length with the end-of-line edge.



FAIL. The constraint applies because the width of the target end-of-line edge is 0.4 (<0.5), the width of the perpendicular edge is 0.5 (<0.7), and the parallel run length is negative. It fails because the spacing between the two edges is 0.2 (<0.3).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minEndOfLineSpacing

```
spacings (
  ( minEndOfLineSpacing tx_layer
    ['horizontal | 'vertical]
    'width f_width ['oppWidth f_oppWidth]
    'distance f_eolWithin ['negativePRL]
    ['paraEdgeWithin f_parWithin 'paraEdgeSpace f_parSpace
      ['paraEdgeCount x_edgeCount]
      ['subtractWidth]
      ['paraMinLength f_parMinLength]
      ['sameMetal | 'sameNet]
      ['nonEolCornerOnly]
      ['prl f_prl]
      ['parallelSameMask]
    ]
    ['cutEolSpace f_cutEolSpace
      ['cutClass {f_width | (f_width f_length) | t_name}]
      ['cutEolAbove | 'cutEolBelow]
      ['enclosureEndWidth g_enclosureEndWidth
        ['enclosureEndWithin g_enclosureEndWithin]
      ]
    ]
    ['widthRanges (g_ranges)]
    ['extendBy f_extendBy]
    ['sizeBy f_sizeBy]
    ['sameMask | 'diffMask]
    ['fillConcaveCorner f_fill]
    ['exactEolWidth]
    ['exceptExactEolWidth (f_exactWidth f_otherWidth)]
    ['wrongDirSpace f_wrongDirSpace]
    ['wrongDirWithin f_wrongDirWithin]
    ['endToEndSpace f_endToEndSpace
      ['otherEndWidth f_otherEndWidth]
      ['oneCutSpace f_oneCutSpace]
      ['twoCutSpace f_twoCutSpace]
      ['extension {f_extension | (f_extension f_wrongDirExtension)}]
      ['bothWires] ['equalRectWidth]
    ]
    ['maxLength f_maxLength | 'minLength f_minLength ['twoSides]]
      ['encloseDistance f_encloseDist 'cutToMetalSpace
f_cutToMetalSpace
      ['above | 'below]
      ['allCuts]
    ]
    ]
    ['exceptExactAligned]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  )
)
```


Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
    ]  
    f_eolSpace  
  )  
); spacings
```

This constraint can also be applied using the following legacy syntax:

```
spacings(  
  ( minEndOfLineSpacing tx_layer  
    (f_width f_eolWithin [f_parWithin [f_parSpace [x_edgeCount]])  
    f_eolSpace)  
  )  
); spacings
```

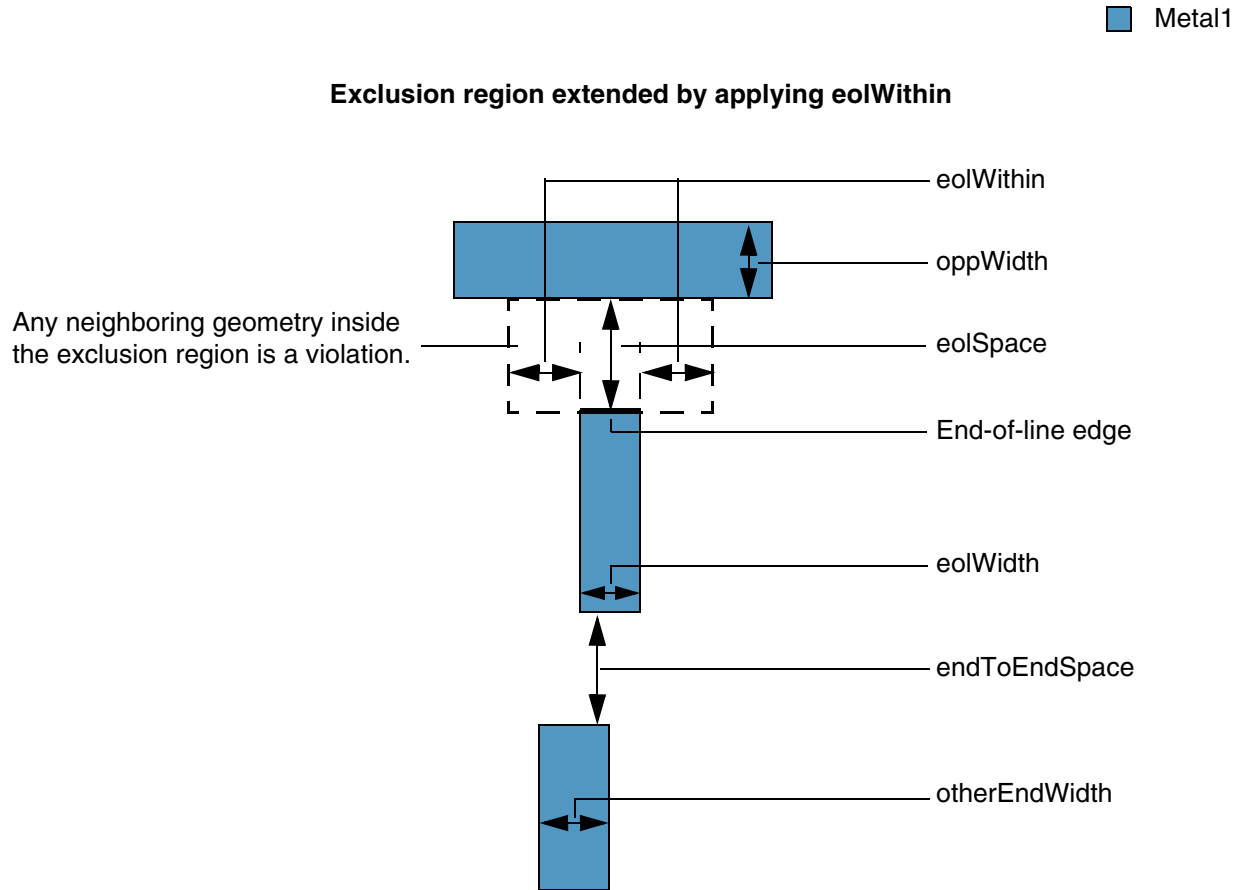
Specifies the minimum edge-to-edge spacing between an end-of-line shape and neighboring parallel shapes that are within the specified region of the end-of-line shape on a layer. The number of supported neighboring parallel shapes is zero, one, or two. Typically, the spacing allowed using this constraint (*eolSpace*) is slightly larger than the minimum allowed spacing on the specified layer. However, the minimum allowed spacing must be greater than the lateral verification distance (*eolWithin*) for the spacing check.

If two face-to-face edges have width less than *eolWidth* and their parallel run length is greater than zero, end-to-end spacing (*endToEndSpace*) applies instead of *eolSpace*.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The following figure illustrates the two scenarios explained above:

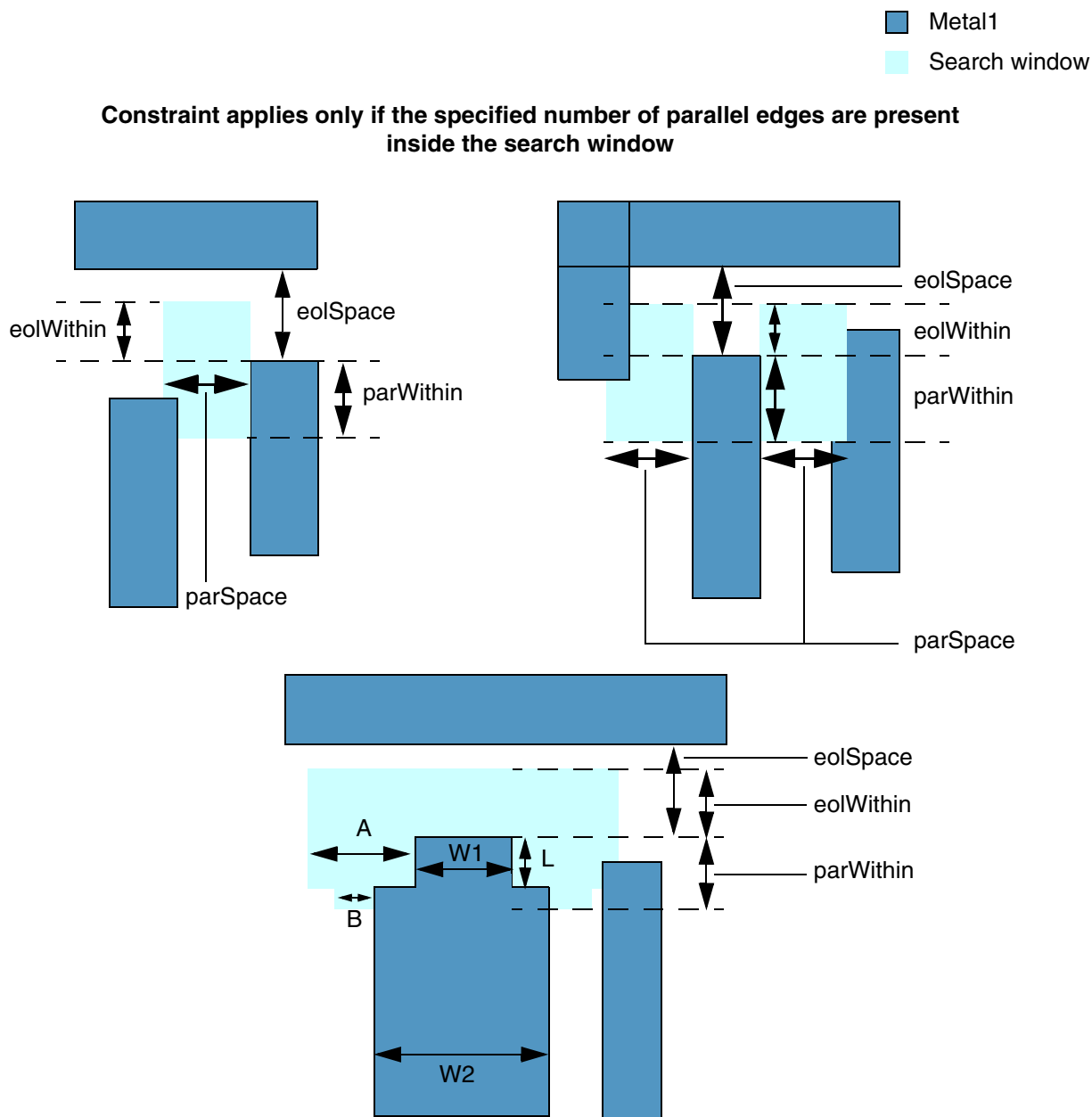


The following figures illustrate how the constraint applies when either one or two parallel edges are present inside the search window defined by *parSpace* and *parWithin* (parallel-edge check). If the number of parallel edges specified is two, they must be present

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

on either side of the end-of-line shape. Edge-to-edge spacing must be greater than or equal to *eolSpace*.



Assuming 'subtractWidth' is specified in this case, $A = \text{parSpace} - W1$, and $B = \text{parSpace} - W2$, that is, the width of the search window is calculated based on the end-of-line width.

L must be greater than or equal to *parMinLength* for the parallel edge on that side to be counted while applying the parallel-edge check.

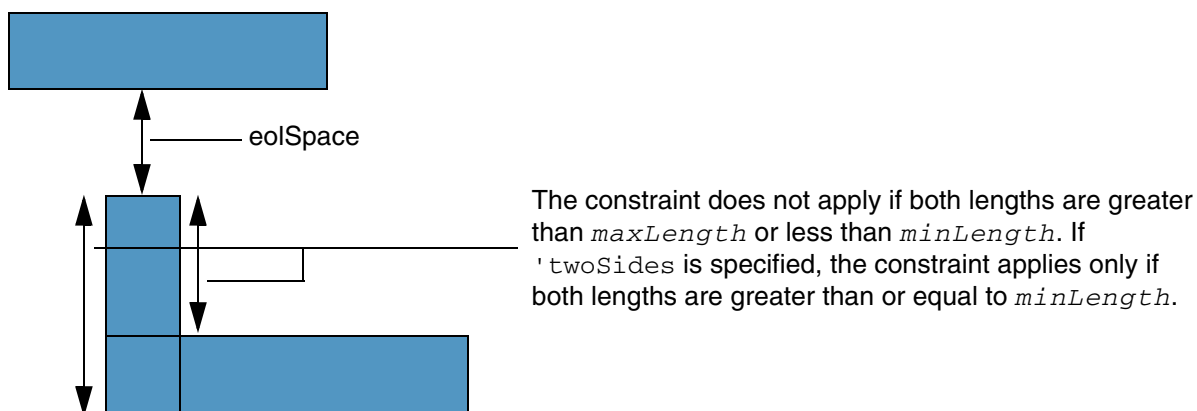
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

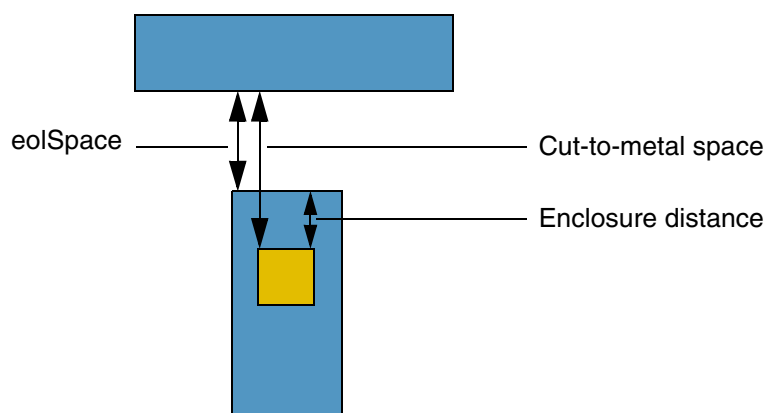
The following figures illustrate two more scenarios in which edge-to-edge spacing must be greater than or equal to *eolSpace*.



Constraint applies only if the minimum and maximum length requirement is satisfied



Constraint applies only if cut-to-metal and cut-to-enclosure-edge spacing requirement is satisfied



Virtuoso Technology Data Constraints Reference

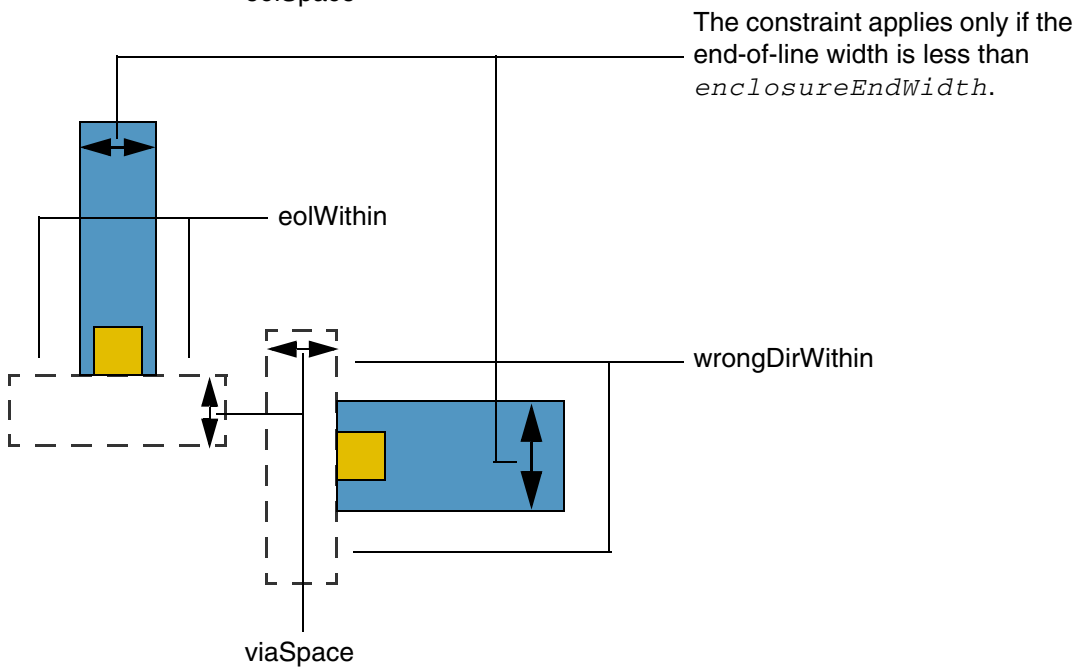
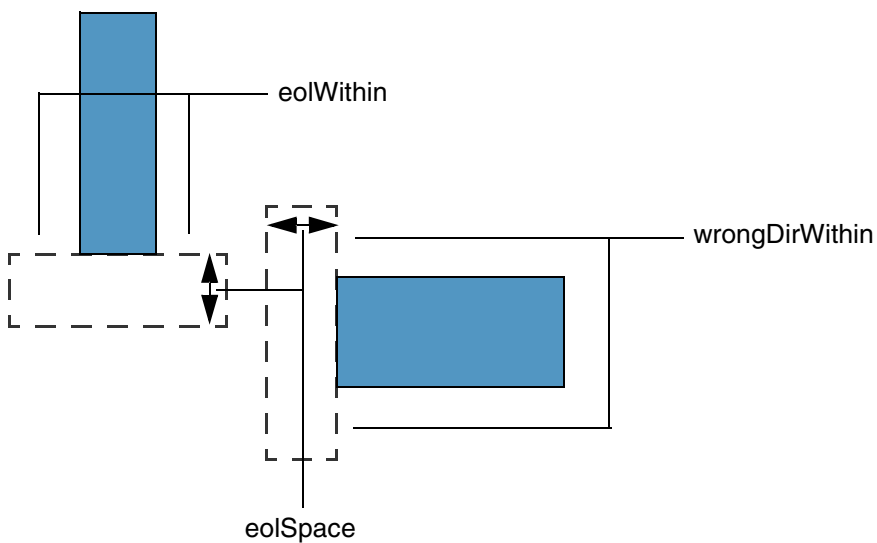
Spacing Constraints (One Layer)

If the two shapes are perpendicular to each other, the lateral verification distance applied in the right and wrong direction is *eolWithin* and *wrongDirWithin*, respectively, as shown below.

↑ The preferred routing direction is vertical and any neighboring geometry inside the exclusion region (depicted using dashed lines) is a violation.

■ Via1
■ Metal1

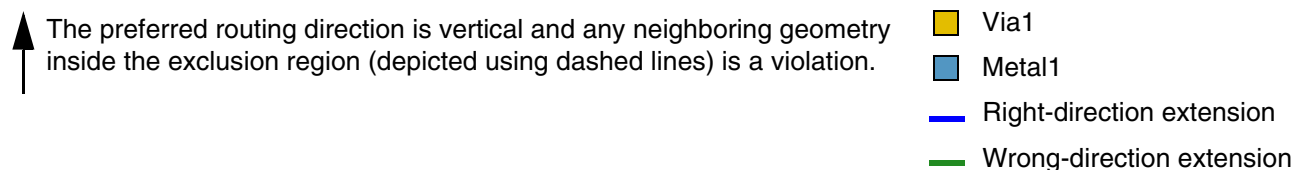
Exclusion region extended by applying *eolWithin* and *wrongDirWithin*



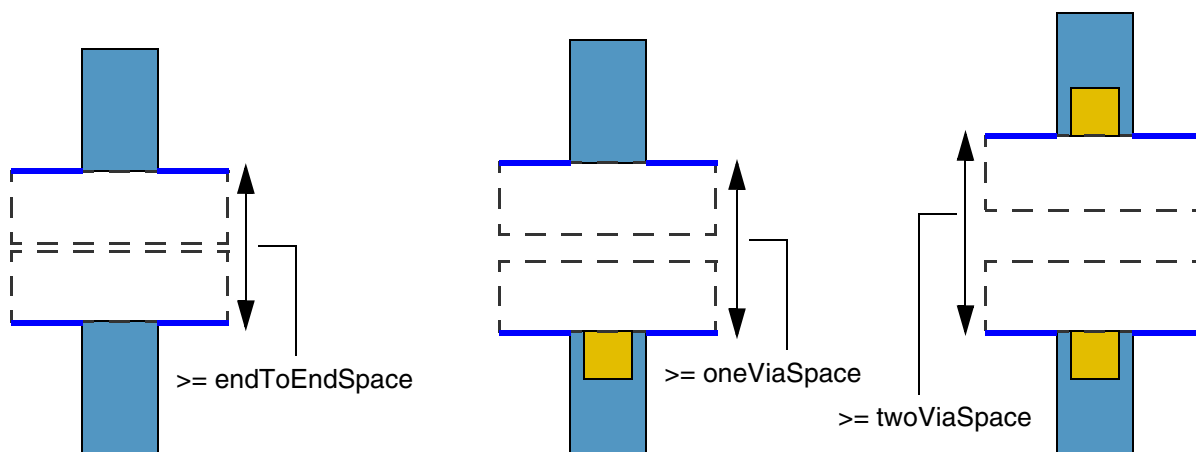
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

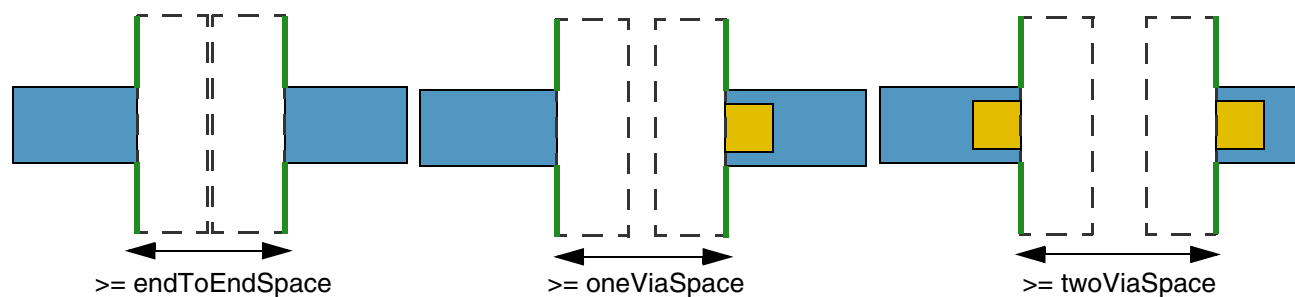
In the following figures, you can see the right-direction and wrong-direction extensions applied to two shapes that are face-to-face. The spacing requirement varies based on the number of vias present.



End-to-end spacing in preferred direction (right direction)



End-to-end spacing in non-preferred direction (wrong direction)



The height of exclusion regions (depicted using dashed lines) is half of the required spacing. An overlap between exclusion regions is a violation.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_eolSpace</i>	The minimum required spacing between an end-of-line edge and its neighboring geometry.

Parameters

<code>'horizontal 'vertical</code>	(Virtuoso Advanced Node for Layout Standard) The direction in which the constraint applies. If direction is not specified, the constraint applies to all end-of-line edges. ■ <code>'horizontal</code>: The constraint applies only to vertical end-of-line edges. ■ <code>'vertical</code>: The constraint applies only to horizontal end-of-line edges. Type: Boolean
<code>'width f_eolWidth</code>	The constraint applies only if the end-of-line width is less than this value.
<code>'oppWidth f_oppWidth</code>	The constraint applies only if a neighboring shape beyond the end-of-line edge has perpendicular span less than this value.
<code>'distance f_eolWithin</code>	The end-of-line edge is extended on both sides by this value to define an exclusion region. A violation occurs if the lateral distance between an end-of-line edge and a neighboring shape is less than this value (lateral distance for applying the spacing check).
<code>'negativePRL</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if a neighboring shape has a parallel run length of less than zero with the shape having an end-of-line edge measured in the routing direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'paraEdgeWithin *f_parWithin*

The constraint applies only if the distance of a neighboring parallel shape from the end-of-line edge, measured along the end-of-line shape, is less than this value (*parWithin*).

'paraEdgeSpace *f_parSpace*

The constraint applies only if the lateral distance between an end-of-line shape and a neighboring parallel shape is less than this value (*parSpace*).

'paraEdgeCount *x_edgeCount*

The constraint applies only if the number of neighboring parallel shapes that satisfy both 'paraEdgeWithin and 'paraEdgeSpace is equal to this value.

Supported values are 0, 1, and 2. If this value is equal to 2, the two neighboring parallel edges must be on opposite sides of the end-of-line shape.

'subtractWidth

The constraint applies only if the lateral distance between an end-of-line shape and a neighboring parallel shape is less than the value obtained after subtracting *eolWidth* from *parSpace*.

Type: Boolean

'paraMinLength *f_parMinLength*

The length of the end-of-line side must be greater than this value for a parallel edge on that side to be counted while applying the parallel-edge check.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'sameNet | 'sameMetal

(Virtuoso Advanced Node for Layout Standard) The connectivity type. If connectivity type is not specified, the constraint applies to any connectivity type, including any two shapes on the same layer with no common metal or net.

- 'sameNet: The constraint applies only if the end-of-line shape and the neighboring shapes are on the same net.
- 'sameMetal: The constraint applies only if the neighboring edges on the side and perpendicular to the end-of-line edge are part of a contiguous same-metal shape.

The constraint does not apply if the neighboring edge on the side extends beyond the search window.

Type: Boolean

'nonEolCornerOnly

The constraint does not apply if a corner inside the search window is formed by a non-end-of-line edge.

Type: Boolean

'prl *f_prl*

The constraint applies only if the parallel run length between an end-of-line side and a neighboring edge is greater than this value.

'parallelSameMask

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the neighboring parallel shape found inside the search window has the same mask as that applied to the end-of-line edge. A neighboring parallel shape with a different mask is ignored.

Type: Boolean

'cutEolSpace *f_cutEolSpace*

The spacing between an end-of-line edge that touches a via cut on a cut layer above or below the metal layer and a non-end-of-line edge perpendicular to it must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'cutEolAbove | 'cutEolBelow`

The `cutEolSpace` spacing is applied only if the metal end-of-line edge touches a shape on a cut layer above or below the metal layer, depending on the parameter specified.

Type: Boolean

`'enclosureEndWidth f_enclosureEndWidth`

The constraint applies only if the width of the end-of-line edge touching a via cut is less than this value.

`'enclosureEndWithin f_enclosureEndWithin`

The end-of-line edge intersecting a via cut is extended on both sides by this value, instead of `eolWithin`, to define the exclusion region.

`'widthRanges (g_ranges)`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the end-of-line width is in the specified range.

Type: Floating-point values specifying a [range](#) of widths that are allowed

`'extendBy f_extendBy`

(Virtuoso Advanced Node for Layout Standard) An extension equal to this value is applied to an end-of-line edge in the projection direction before the constraint is applied. The extension is applied to both shapes, and not just to the end-of-line shape.

`'sizeBy f_sizeBy`

(Virtuoso Advanced Node for Layout Standard) An end-of-line edge is extended on both sides by this value before the constraint is applied.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'sameMask | 'diffMask

(Virtuoso Advanced Node for Layout Standard) The mask type.

- 'sameMask: The constraint applies only between shapes on the same mask.
- 'diffMask: The constraint applies only between shapes on different masks.

Type: Boolean

'fillConcaveCorner *f_fill*

(Virtuoso Advanced Node for Layout Only) A concave corner that is in the neighborhood of an end-of-line edge is filled with a triangular shape whose sides measured along the concave edges are equal to this value before the spacing between the concave corner and the end-of-line edge is measured.

'exactEolWidth

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the end-of-line width is equal to this value.

Type: Boolean

'exceptExactEolWidth (*f_exactWidth f_otherWidth*)

(Virtuoso Advanced Node for Layout Only) The constraint does not apply if the end-of-line width is equal to *exactWidth* and the width of the neighboring edge is less than or equal to *otherWidth*.

'wrongDirSpace *f_wrongDirSpace*

(Virtuoso Advanced Node for Layout Standard) The edge-to-edge spacing measured in the wrong direction (non-preferred routing direction) must be greater than or equal to this value.

'wrongDirWithin *f_wrongDirWithin*

An end-of-line edge of a shape in the non-preferred routing direction is extended on both sides by this value, instead of *eolWithin*, to form the exclusion region. When this parameter is specified, the *eolWithin* value applies only to the end-of-line edges of shapes in the preferred routing direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'endtoEndSpace f_endToEndSpace`

The spacing between two end-of-line edges that face each other and have parallel run length greater than zero must be greater than or equal to this value.

`'otherEndWidth f_otherEndWidth`

The constraint applies only if a shape facing an end-of-line edge has width less than this value. This parameter can be specified with or without `endtoEndSpace` parameter. If it is specified with `endtoEndSpace` parameter with the value greater than `minEndOfLineSpacing` constraint value, the DRD checker checks the width of other end of the shapes in the region created between `minEndOfLineSpacing` and `endToEndSpace`. If the `endToEndSpace` parameter is not specified, `otherEndWidth` is checked verifying the region created for `minEndOfLineSpacing` constraint.

`'oneCutSpace f_oneCutSpace`

The spacing between two end-of-line edges, one of which touches a via cut, must be greater than or equal to this value.

`'twoCutSpace f_twoCutSpace`

The spacing between two end-of-line edges, both of which touch a via cut each, must be greater than or equal to this value.

`'extension {f_extension | (f_extension f_wrongDirExtension)}`

- *f_extension*: The extension applied to both ends of an end-of-line edge of a shape routed in the preferred direction. For example, if the preferred routing direction is vertical, an end-of-line shape that is routed vertically has a horizontal end-of-line edge and horizontal extensions (right-direction extensions).
- *f_wrongDirExtension*: The extension applied to both ends of an end-of-line edge of a shape routed in the non-preferred direction. For example, if the preferred routing direction is vertical, an end-of-line shape that is routed horizontally has a vertical end-of-line edge and vertical extensions (wrong-direction extensions).

`'maxLength f_maxLength`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The constraint does not apply if both sides of the end-of-line shape are greater than this value in length.

'minLength *f_minLength*

The constraint does not apply if both sides of the end-of-line shape are less than this value in length.

'twoSides

The constraint applies only if both sides of the end-of-line shape are greater than or equal to *minLength*.

Type: Boolean

'bothWires

(Virtuoso Advanced Node for Layout Standard) The spacing between two end-of-line edges must be greater than or equal to *endToEndSpace* if the length of both the shapes is greater than or equal to *minLength* or less than or equal to *maxLength*, depending on which of the two parameters, 'minLength or 'maxLength, is specified.

Type: Boolean

'equalRectWidth

The constraint applies only if the length of the end-of-line edge is equal to the width of the shape. If multiple *minEndOfLineSpacing* constraints are defined on a layer and 'equalRectWidth is specified with even one of them, then 'equalRectWidth must be specified with all of them.

Type: Boolean

'encloseDistance *f_encloseDistance*

The constraint applies only if there is a via cut above or below the metal layer and the distance between the via-cut edge and the end-of-line edge is less than this value.

'cutToMetalSpace *f_cutToMetalSpace*

The constraint applies only if the distance between the via-cut edge and the edge facing the end-of-line edge is less than this value.

'above | 'below

The constraint applies to a shape on a cut layer either above or below the metal layer. If this parameter is not specified, the constraint applies to shapes on cut layers both above and below the metal layer.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'allCuts` The constraint applies to all cuts that connect a same-metal shape, both above and below the metal layer. Otherwise, only one such via cut needs to satisfy the constraint.

Type: Boolean

`'exceptExactAligned`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the shapes are not exactly aligned.

Type: Boolean

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Examples

- Example 1: minEndOfLineSpacing with widthRanges and exactEolWidth
- Example 2: minEndOfLineSpacing with oppWidth and minLength
- Example 3: minEndOfLineSpacing with paraEdgeWithin, paraEdgeSpace, and subtractWidth
- Example 4: minEndOfLineSpacing with endToEndSpace, minLength, and twoSides
- Example 5: minEndOfLineSpacing with widthRanges, extendBy, and sizeBy
- Example 6: minEndOfLineSpacing with widthRanges, extendBy, sizeBy, and endToEndSpace
- Example 7: minEndOfLineSpacing with paraEdgeWithin, paraEdgeSpace, and sameMetal
- Example 8: minEndOfLineSpacing with diffMask, endToEndSpace, and exceptExactAligned
- Example 9: minEndOfLineSpacing with fillConcaveCorner
- Example 10: minEndOfLineSpacing with exceptExactEolWidth
- Example 11: minEndOfLineSpacing with paraEdgeWithin, paraEdgeSpace, and nonEolCornerOnly
- Example 12: minEndOfLineSpacing with equalRectWidth
- Example 13: minEndOfLineSpacing with cutEolSpace, wrongDirWithin, and endToEndSpace
- Example 14: minEndOfLineSpacing with horizontal
- Example 15: minEndOfLineSpacing with paraEdgeWithin, paraEdgeSpace, paraMinLength, encloseDistance, and cutToMetalSpace
- Example 16: minEndOfLineSpacing with width, eolWithin, parWithin, parSpace, and edgeCount

Virtuoso Technology Data Constraints Reference

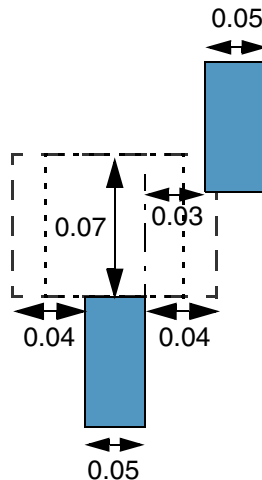
Spacing Constraints (One Layer)

Example 1: minEndOfLineSpacing with widthRanges and exactEolWidth

The end-of-line spacing between Metal1 wires must be at least 0.09 if the end-of-line width is greater than or equal to 0.05 and less than 0.1, and at least 0.07 if the end-of-width is equal to 0.1.

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.051
    'distance 0.02
    'widthRanges ("[0.05 0.10)")
    0.09
  )
  ( minEndOfLineSpacing "Metal1"
    'width 0.1
    'distance 0.04
    'exactEolWidth
    0.07
  )
) ;spacings
```

■ Metal1



The first `minEndOfLineSpacing` constraint does not apply because the neighboring shape is at a distance of 0.03 (>0.02) from the end-of-line edge. The second `minEndOfLineSpacing` constraint does not apply because the end-of-line width is 0.05.

Virtuoso Technology Data Constraints Reference

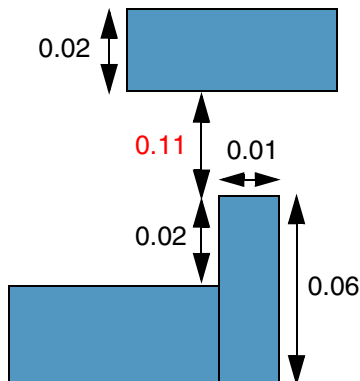
Spacing Constraints (One Layer)

Example 2: *minEndOfLineSpacing* with *oppWidth* and *minLength*

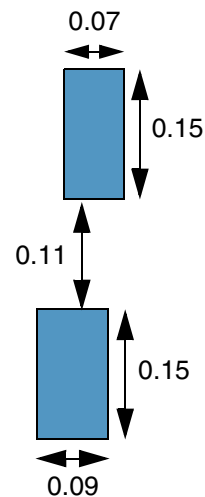
The end-of-line spacing between Metal1 wires must be at least 0.12 when the minimum length of an end-of-line side is less than 0.06 and the width of the wire perpendicular to the end-of-line edge is 0.08. The lateral verification distance for the spacing check is 0.05.

```
spacings (  
  ( minEndOfLineSpacing "Metal1"  
    'width 0.1 'oppWidth 0.08  
    'distance 0.05  
    'minLength 0.06  
    0.12  
  )  
) ;spacings
```

■ Metal1



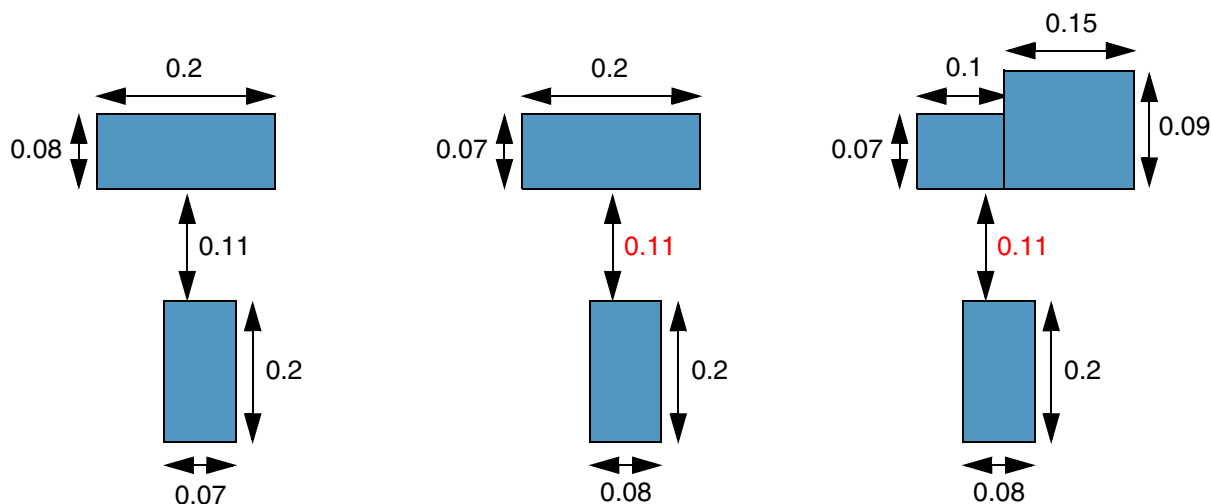
a) FAIL. The end-of-line width is 0.01 (<0.1), the length of an end-of-line side is 0.02, and the width of the wire perpendicular to the end-of-line edge is 0.02 (<0.08). However, the end-of-line spacing is 0.11 (<0.12). The constraint would not apply if 'twoSides' was specified.



b) The constraint does not apply because the opposite wire has a perpendicular span of 0.15 (>0.08) to the end-of-line edge.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply because the width of the wire perpendicular to the end-of-line edge is 0.08.

d) FAIL. The width of the wire perpendicular to the end-of-line edge is 0.07 (< 0.08). However, the end-of-line spacing is 0.11 (< 0.12).

e) FAIL. A portion of the opposite wire (left segment) is 0.07 wide (< 0.08). However, the end-of-line spacing is 0.11 (< 0.12).

Example 3: *minEndOfLineSpacing* with *paraEdgeWithin*, *paraEdgeSpace*, and *subtractWidth*

The end-of-line spacing between Metal1 wires must be at least 0.15 if the following conditions are met:

- The end-of-line width is less than 0.1.
- A parallel neighboring edge is present inside the search window defined by *eolWithin* = 0.05, *parWithin* = 0.03, and *parSpace* = 0.16. The dimensions of the search window are calculated dynamically based on the end-of-line width of the wire.

Virtuoso Technology Data Constraints Reference

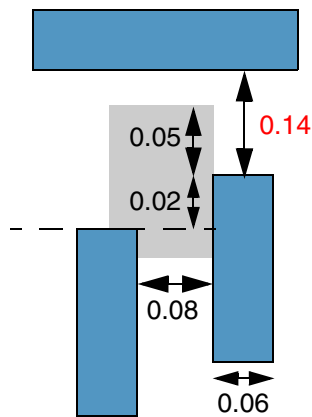
Spacing Constraints (One Layer)

```

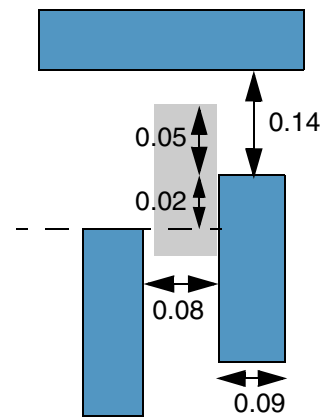
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.1
    'distance 0.05
    'paraEdgeWithin 0.03 'paraEdgeSpace 0.16
    'subtractWidth
    0.15
  )
) ;spacings

```

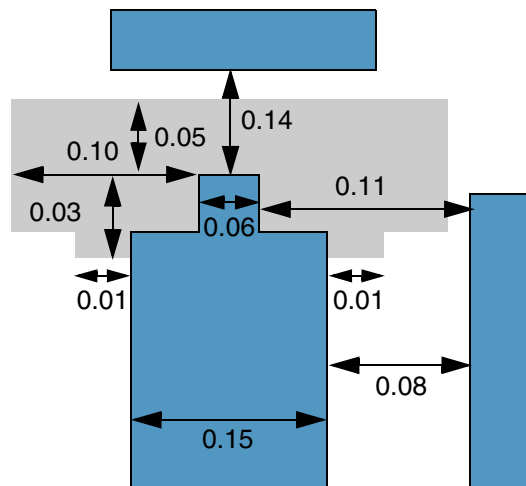
Metal1
 Search window



a) FAIL. A neighboring parallel edge is present inside the search window—when the end-of-line width is 0.06, the width of the search window is 0.1 ($=0.16-0.06$). However, the end-of-line spacing is 0.14 (<0.15).



b) The constraint does not apply because the neighboring parallel edge does not lie within the search window—when the end-of-line width is 0.09, the width of the search window is 0.07 ($=0.16-0.09$).



c) The constraint does not apply because the neighboring parallel edge does not lie within the search window. The dimensions of the search window are calculated dynamically based on the dimensions of the end-of-line shape; for example, where the end-of-line width is 0.06, the search window is 0.1 wide ($=0.16-0.06$), and where the end-of-line width is 0.15, the search window is 0.01 wide ($=0.16-0.15$).

Virtuoso Technology Data Constraints Reference

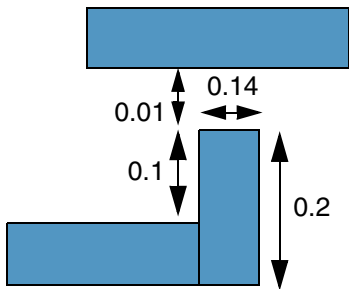
Spacing Constraints (One Layer)

Example 4: *minEndOfLineSpacing* with *endToEndSpace*, *minLength*, and *twoSides*

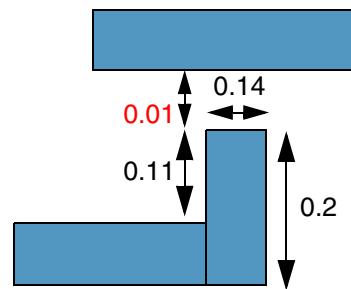
The end-of-line spacing between Metal1 wires must be at least 0.1 and the end-to-end spacing must be at least 0.12 if the length of both end-of-line sides is greater than or equal to 0.11.

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.15
    'distance 0.05
    'endToEndSpace 0.12
    'minLength 0.11 'twoSides
    0.1
  )
) ;spacings
```

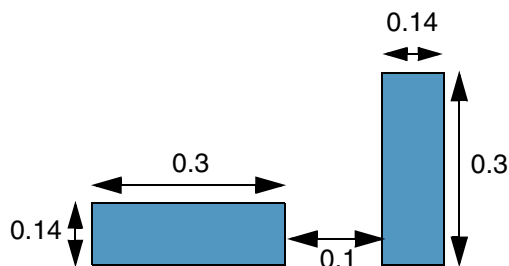
■ Metal1



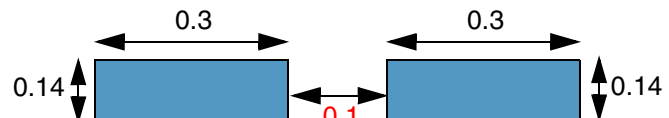
a) The constraint does not apply because only the right end-of-line side is longer than 0.11. A violation would occur if 'twoSides was not specified.



b) FAIL. Both end-of-line sides are greater than or equal to 0.11 long. However, the end-of-line spacing is 0.01 (<0.1).



c) PASS. The end-of-line spacing is 0.1. End-to-end spacing is ignored because the two end-of-line edges do not face each other.



d) FAIL. The two end-of-line edges face each other and have parallel run length greater than zero. However, end-to-end spacing is only 0.1 (<0.12).

Virtuoso Technology Data Constraints Reference

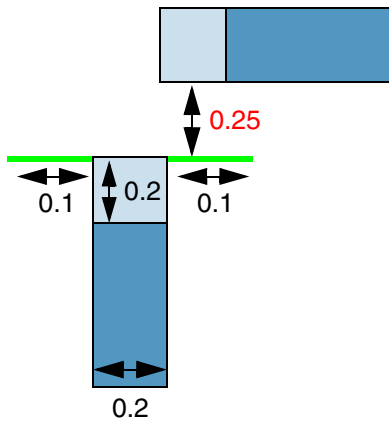
Spacing Constraints (One Layer)

Example 5: *minEndOfLineSpacing* with *widthRanges*, *extendBy*, and *sizeBy*

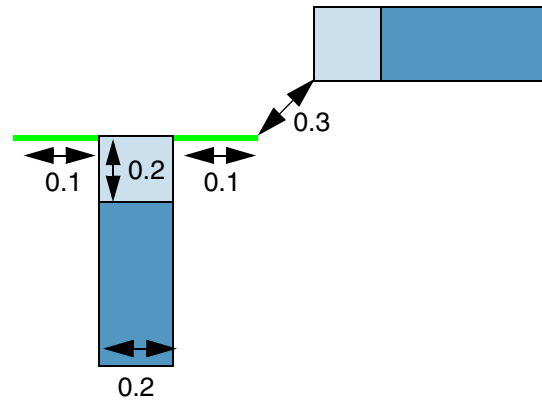
The end-of-line spacing must be at least 0.3 when measured in the vertical direction if the end-of-line edge has width less than or equal to 0.3 or equal to 0.6 or to 0.8. The spacing is measured after applying an extension of 0.2 to the end-of-line edge in the projection direction (*extendBy*) and 0.1 on both sides of the end-of-line edge (*sizeBy*).

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'vertical
    'width 0.31
    'distance 0.1
    'widthRanges ("<=0.3" 0.6 0.8)
    'extendBy 0.2
    'sizeBy 0.1
    0.3
  )
) ;spacings
```

■ Metal1
■ Metal1 extension
— Sized by



a) FAIL. The width of the end-of-line (0.2) is within the specified range (≤ 0.3), but the end-of-line spacing is 0.25 (< 0.3).



b) PASS. The width of the end-of-line (0.2) is within the specified range (≤ 0.3) and the end-of-line spacing is 0.3.

Virtuoso Technology Data Constraints Reference

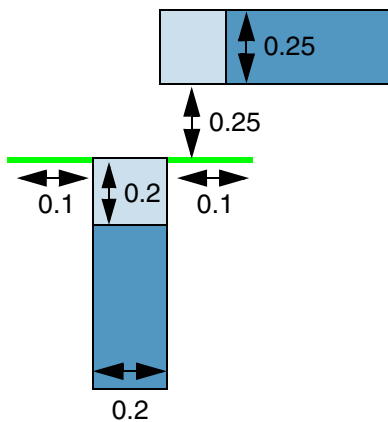
Spacing Constraints (One Layer)

Example 6: *minEndOfLineSpacing* with *widthRanges*, *extendBy*, *sizeBy*, and *endToEndSpace*

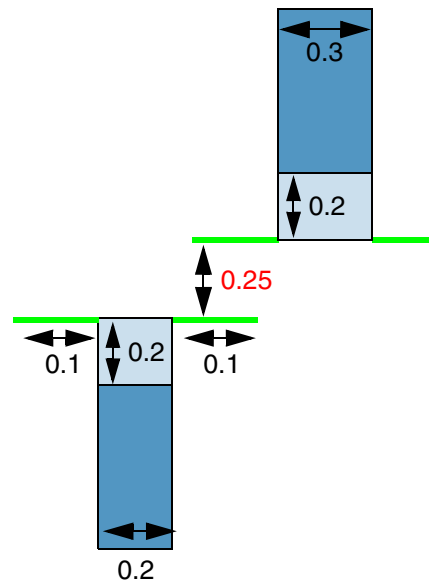
The end-of-line spacing must be less than 0.3 and end-to-end spacing must be equal to 0.3 if the end-of-line edge has width less than or equal to 0.3 or equal to 0.6 or to 0.8. The spacing is measured after applying an extension of 0.2 to the end-of-line edge in the projection direction (*extendBy*) and 0.1 on both sides of the end-of-line edge (*sizeBy*).

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.31
    'distance 0.1
    'widthRanges ("<=0.3" "0.6" "0.8")
    'extendBy 0.2
    'sizeBy 0.1
    'endToEndSpace 0.3
    0.3
  )
) ;spacings
```

 Metal1
 Metal1 extension
 Sized by



a) FAIL. The neighboring shape is within the region created by constraint value (0.3) and other parameters such as distance, extendBy and sizeBy. The width conditions are also met.



b) FAIL. The two end-of-line edges (width ≤ 0.3) face each other and have parallel run length greater than zero when extended laterally. However, the end-to-end spacing is 0.25 (< 0.3).

Virtuoso Technology Data Constraints Reference

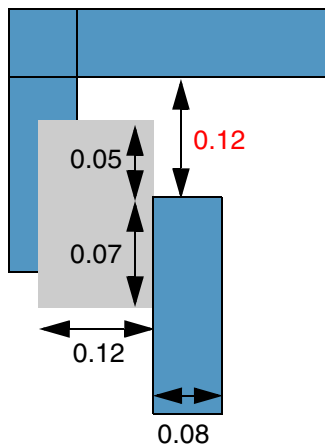
Spacing Constraints (One Layer)

Example 7: *minEndOfLineSpacing* with *paraEdgeWithin*, *paraEdgeSpace*, and *sameMetal*

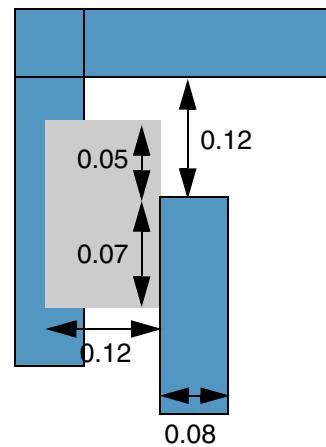
The end-of-line spacing between Metal1 wires must be at least 0.13 if a parallel edge present inside the search window is part of a contiguous same-metal shape. The search window is defined by *eolWithin* = 0.05, *parWithin* = 0.07, and *parSpace* = 0.12.

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.1
    'distance 0.05
    'paraEdgeWithin 0.07 'paraEdgeSpace 0.12
    'sameMetal
    0.13
  )
) ;spacings
```

■ Metal1
■ Search window



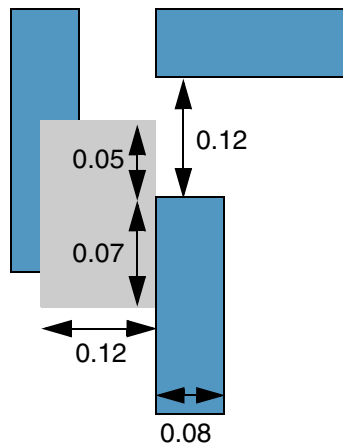
a) FAIL. The neighboring edges on the left and on top belong to a contiguous same-metal shape. However, the end-of-line spacing is 0.12 (<0.13).



b) The constraint does not apply because the neighboring left edge covers the entire length of the search window.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply because neighboring shapes do not form a contiguous same-metal shape. A violation would occur if 'sameMetal' was not specified.

Virtuoso Technology Data Constraints Reference

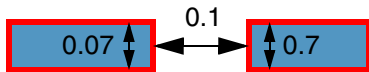
Spacing Constraints (One Layer)

Example 8: *minEndOfLineSpacing* with *diffMask*, *endToEndSpace*, and *exceptExactAligned*

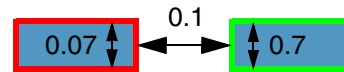
The end-of-line spacing must be at least 0.11 and end-to-end spacing must be at least 0.13 between non-exactly-aligned wires on different masks if the end-of-line edge has width less than 0.08. The lateral verification distance for the spacing check is 0.02.

```
spacings (  
  ( minEndOfLineSpacing "Metal1"  
    'width 0.08  
    'distance 0.02  
    'diffMask  
    'endToEndSpace 0.13  
    'exceptExactAligned  
    0.11  
  )  
) ;spacings
```

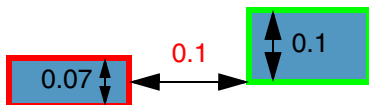
■ Metal1
■ mask1
■ mask2



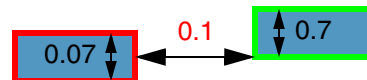
a) The constraint does not apply because both shapes are on the same mask.



b) The constraint does not apply because the two shapes are exactly aligned.



c) FAIL. The end-of-line width is 0.07 (<0.08), and the shapes are on different masks and are not exactly aligned. However, the end-of-line spacing is only 0.1 (<0.11).



d) FAIL. The end-of-line width of both shapes is 0.07 (<0.08), and the shapes are on different masks and are not exactly aligned. However, the end-to-end spacing is only 0.1 (<0.13).

Virtuoso Technology Data Constraints Reference

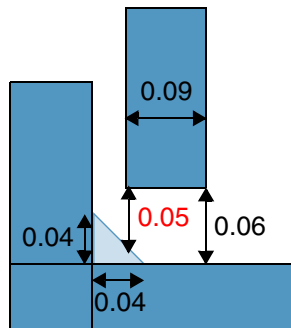
Spacing Constraints (One Layer)

Example 9: *minEndOfLineSpacing* with *fillConcaveCorner*

A concave corner present in the neighborhood of an end-of-line edge must be filled by a triangle whose sides along the concave edges are equal to 0.04 before end-of-line spacing of 0.06 is applied. The end-of-line width of the wire must be less than 0.1.

```
spacings (  
  ( minEndOfLineSpacing "Metal1"  
    'width 0.1  
    'distance 0  
    'fillConcaveCorner 0.04  
    0.06  
  )  
); spacings
```

■ Metal1



FAIL. After the concave corner is filled by a triangular shape with sides equal to 0.04 along the concave edges, the end-of-line spacing is 0.05 (< 0.06).

Virtuoso Technology Data Constraints Reference

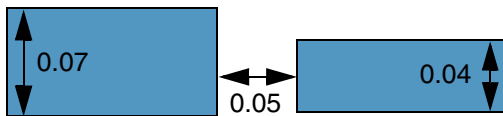
Spacing Constraints (One Layer)

Example 10: *minEndOfLineSpacing* with *exceptExactEolWidth*

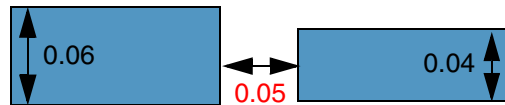
The end-of-line spacing between Metal1 wires must be at least 0.06 except if the end-of-line width of one shape is equal to 0.07 and the end-of-line width of the other shape is less than 0.05.

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.1
    'distance 0
    'exceptExactEolWidth (0.07 0.05)
    0.06
  )
) ;spacings
```

■ Metal1



a) The constraint does not apply because the width of the left shape is 0.07 and the width of the right shape is less than 0.05.



b) FAIL. The width of the left shape is less than 0.07. However, the end-of-line spacing is 0.05 (<0.06).

Example 11: *minEndOfLineSpacing* with *paraEdgeWithin*, *paraEdgeSpace*, and *nonEolCornerOnly*

The end-of-line spacing between Metal1 wires must be at least 0.13 if the following conditions are met:

- The end-of-line width is less than 0.1.
- A corner inside the search window is formed by another end-of-line edge.
- A parallel edge is present inside the search window defined by *eolWithin* = 0.05, *parWithin* = 0.07, and *parSpace* = 0.12.

Virtuoso Technology Data Constraints Reference

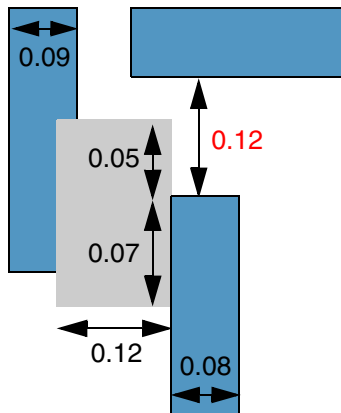
Spacing Constraints (One Layer)

```

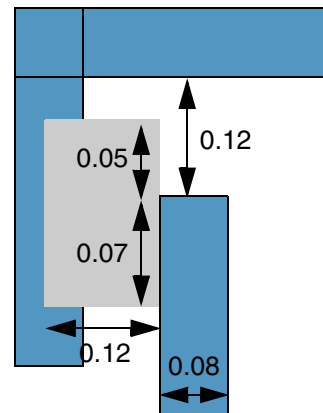
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.1
    'distance 0.05
    'paraEdgeWithin 0.07 'paraEdgeSpace 0.12
    'nonEolCornerOnly
    0.13
  )
) ;spacings

```

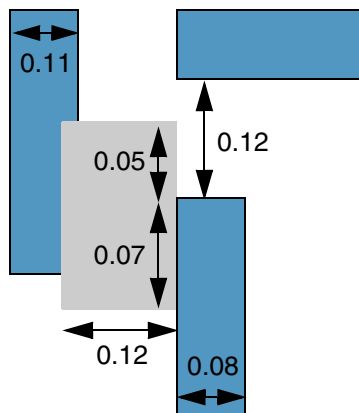
Metal1
 Search window



a) FAIL. A corner that is formed by another end-of-line edge is present inside the search window. However, the end-of-line spacing is 0.12 (<0.13).



b) The constraint does not apply because the neighboring shape does not have a corner inside the search window.



c) The constraint does not apply because the corner inside the search window is not formed by an end-of-line edge (the end-of-line width of the neighboring shape is 0.11, which is greater than 0.1).

Virtuoso Technology Data Constraints Reference

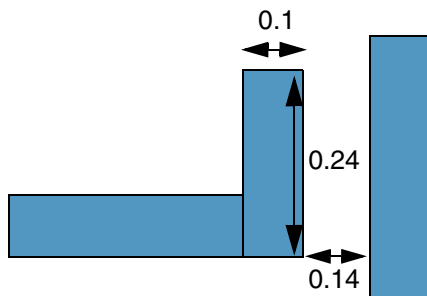
Spacing Constraints (One Layer)

Example 12: *minEndOfLineSpacing* with *equalRectWidth*

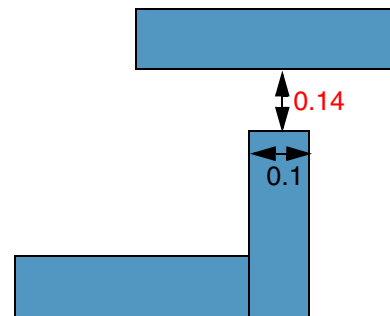
The end-of-line spacing between Metal1 wires must be at least 0.15 if the end-of-line width is equal to the width of the wire. The end-of-line width must be less than 0.25 and the lateral verification distance for the spacing check is 0.10.

```
spacings (  
  ( minEndOfLineSpacing "Metal1"  
    'width 0.25  
    'distance 0.10  
    'equalRectWidth  
    0.15  
  )  
); spacings
```

■ Metal1



a) The constraint does not apply because the end-of-line width (0.24) is greater than the width of the shape (0.1).



b) FAIL. The end-of-line width is 0.1, which is equal to the width of the shape. However, the end-of-line spacing is 0.14 (< 0.15).

Example 13: minEndOfLineSpacing with cutEolSpace, wrongDirWithin, and endToEndSpace

- The end-of-line spacing between Metal1 wires must be at least 0.1 if the following conditions are met:
 - The end-of-line width is less than 0.08.
 - The lateral verification distance for the spacing check is 0.05 for a shape routed in the preferred routing direction (*eolWithin*).
 - The lateral verification distance for the spacing check is 0.06 for a shape routed in the non-preferred direction (*wrongDirWithin*).
- The required edge-to-edge spacing between Metal1 wires must be at least 0.12 if the following conditions are met:
 - The end-of-line edge intersects a VA via cut on a cut layer either above or below the metal layer.
 - The end-of-line width is less than 0.14.
 - The lateral verification distance for the spacing check is 0.05.
- The end-to-end spacing between Metal1 wires must be greater than or equal 0.11 if the end-of-line edges do not touch a via cut, greater than or equal to 0.13 if an end-of-line edge touches a via cut, and greater than or equal to 0.15 if both end-of-line edges touch a via cut each. The right direction extension is 0.03 and the wrong direction extension is 0.08.

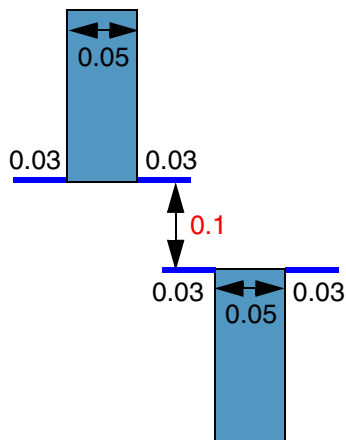
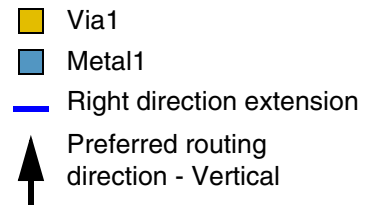
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

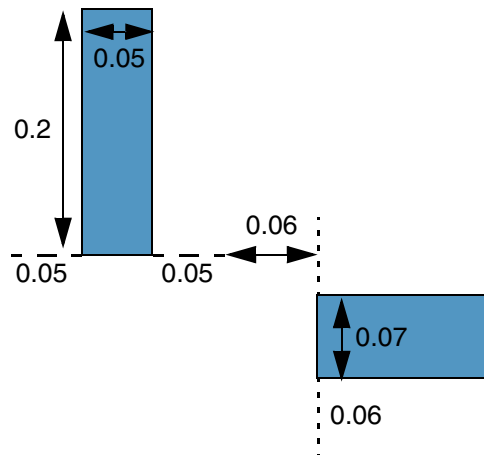
```

spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.08
    'distance 0.05
    'cutEolSpace 0.12
    'cutClass "VA"
    'enclosureEndWidth 0.14
    'enclosureEndWithin 0.05
    'wrongDirWithin 0.06
    'endToEndSpace 0.11
    'oneCutSpace 0.13 'twoCutSpace 0.15
    'extension 0.03 0.08
    0.1
  )
) ;spacings

```



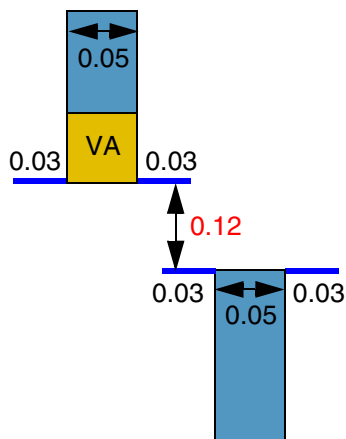
a) FAIL. The end-to-end edges are both less than 0.08 wide and have a parallel run length greater than zero after an extension of 0.03 is applied to the edges in the non-preferred routing direction. However, the end-to-end spacing is only 0.1 (<0.11).



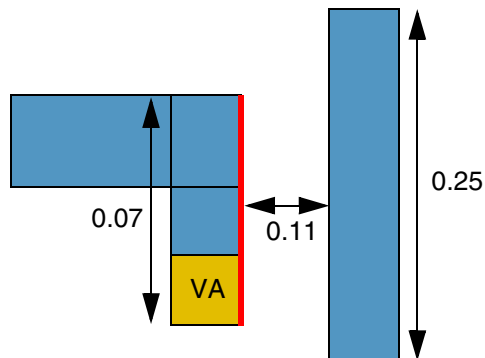
b) PASS. The end-of-line spacing is 0.11 (0.05+0.06), which is greater than the required value of 0.1. (The dashed lines indicate the lateral verification distance in the non-preferred routing direction (*eolWithin*) and the dotted lines indicate the lateral verification distance in the preferred routing direction (*wrongDirWithin*).)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) FAIL. The via cut touches an end-of-line edge. Therefore, *oneCutSpace* spacing of 0.13 is required. However, the spacing between the two end-of-line edges is 0.12 (<0.13).



d) PASS. The end-of-line spacing is 0.11 (>0.1). (The red edge is an end-of-line edge (<0.08), and not an enclosure end-of-line edge. Therefore, the cut-to-end-of-line spacing of 0.12 does not apply.)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

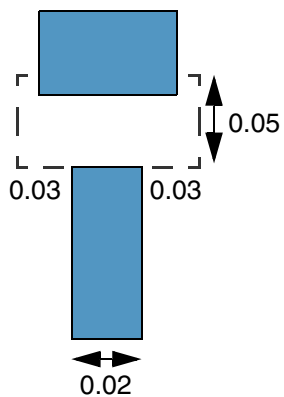
Example 14: *minEndOfLineSpacing* with horizontal

The end-of-line spacing between Metal1 wires must be at least 0.05 when measured in the horizontal direction if the following conditions are met:

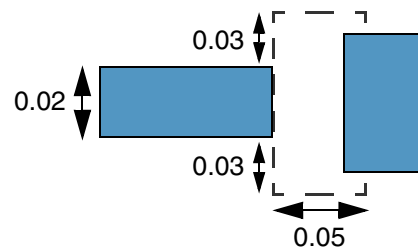
- The end-of-line width is less than 0.03.
- The lateral verification distance for the spacing check is 0.03.

```
spacings (  
  ( minEndOfLineSpacing "Metal1"  
    'width 0.03  
    'distance 0.03  
    'horizontal  
    0.05  
  )  
) ;spacings
```

■ Metal1



a) The constraint does not apply because the end-of-line spacing is measured vertically from a horizontal end-of-line edge.



b) FAIL. The end-of-line spacing is measured horizontally, but is less than the required value (0.05).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

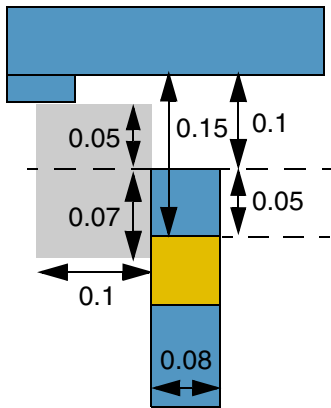
Example 15: *minEndOfLineSpacing* with *paraEdgeWithin*, *paraEdgeSpace*, *paraMinLength*, *encloseDistance*, and *cutToMetalSpace*

The end-of-line spacing between Metal1 wires must be at least 0.15 if the following conditions are met:

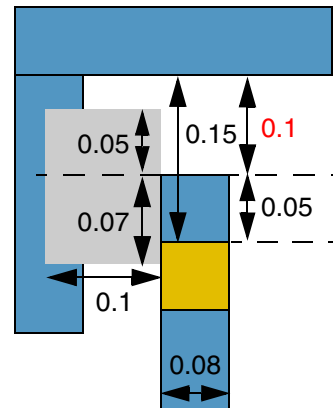
- The width of the end-of-line shape is less than 0.1 (*eolWidth*) and the length of the end-of-line side is greater than or equal to 0.1 (*paraMinLength*).
- A parallel edge is present inside the search window defined by *eolWithin* = 0.05, *parWithin* = 0.07, and *parSpace* = 0.1.
- The distance of a via cut below the metal layer to the end-of-line edge is less than 0.06 (enclosed distance) and to an edge beyond the end-of-line edge is less than 0.16 (cut-to-metal space).

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.1
    'distance 0.05
    'paraEdgeWithin 0.07 'paraEdgeSpace 0.1
    'paraMinLength 0.1
    'encloseDistance 0.06 'cutToMetalSpace 0.16
    'below
    0.15
  )
) ;spacings
```

Via1
 Metal1
 Search window



a) The constraint does not apply because no parallel edge is present inside the search window.



b) FAIL. The enclosed distance is 0.05 (<0.06) and the cut-to-metal space is 0.15 (<0.16). Additionally, the neighboring edge meets the criteria specified for a parallel edge. However, the end-of-line spacing is 0.01 (<0.15).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 16: *minEndOfLineSpacing* with *width*, *eolWithin*, *parWithin*, *parSpace*, and *edgeCount*

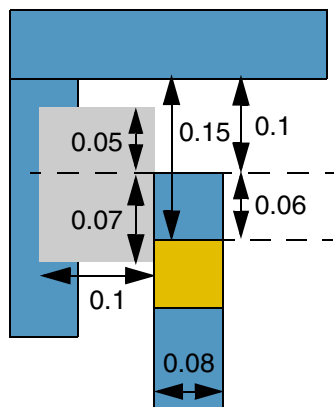
The end-of-line spacing, or *eolSpace*, between Metal1 wires must be at least 0.43 if:

- *width* is 0.11
- *eolWithin* is 0.12
- *parWithin* is 0.13
- *parSpace* is 0.14
- *edgeCount* is 15

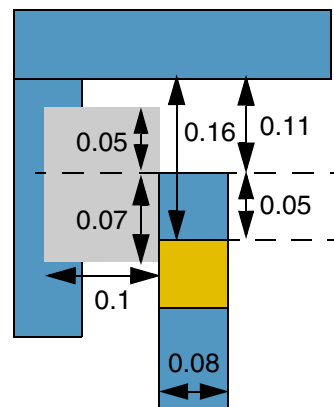
```
spacings (  
  ( minEndOfLineSpacing "Metal1"  
    (0.11 0.12 0.13 0.14 15 0.43)  
  ) ;spacings
```

Virtuoso Technology Data Constraints Reference

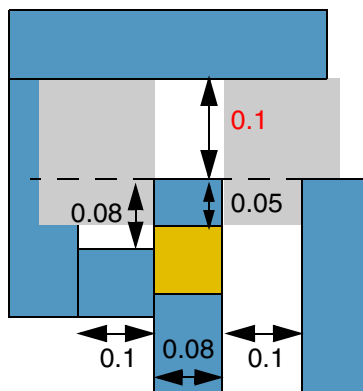
Spacing Constraints (One Layer)



c) The constraint does not apply because the enclosed distance is 0.06. For the constraint to apply, the enclosed distance must be less than 0.06.



d) The constraint does not apply because the cut-to-metal space is 0.16. For the constraint to apply, the cut-to-metal space must be less than 0.16.



e) FAIL. The minimum length of the left side of the end-of-line shape is 0.08 (< 0.1), which means the parallel edge on the left is ignored. However, a parallel edge is present inside the search window on the right. Therefore, the end-of-line spacing must be at least 0.15.

Virtuoso Technology Data Constraints Reference

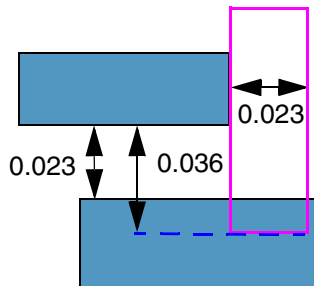
Spacing Constraints (One Layer)

Example 17: *minEndOfLineSpacing* with *distance* and *negativePRL*

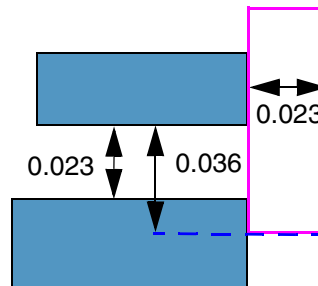
The end-of-line spacing must be at least 0.023 with other parameters as indicated. The constraint applies only if the parallel run length measured in the routing direction is negative.

```
spacings (
  ( minEndOfLineSpacing "Metal1"
    'width 0.054
    'distance 0.036 'negativePRL
    'endToEndSpacing 0.026
    'minLength 0.02 'twoSides
    0.023
  )
) ;spacings
```

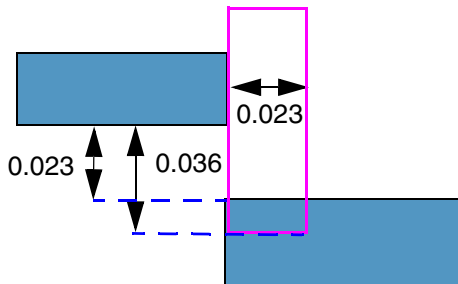
■ Metal1
□ Search window



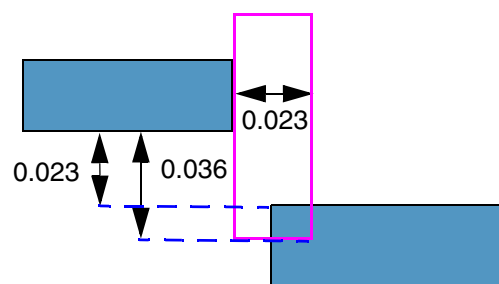
PASS. The constraint does not apply because the parallel run length between the two shapes is more than 0.



PASS. The constraint does not apply. The parallel run length is not less than 0 because the edges are coincident.



PASS. The constraint does not apply. The parallel run length is not less than 0 because the edges are coincident.



FAIL. The constraint applies because the parallel run length is negative and the spacing is violated.

minEndOfLineToConcaveCornerSpacing

```
spacings (
    ( minEndOfLineToConcaveCornerSpacing tx_layer
      'width f_width
      ['minLength f_minLength]
      ['dualAdjacentLength (f_minAdjLength1 f_minAdjLength2)]
      f_spacing
    )
); spacings
```

Specifies the minimum spacing between an end-of-line edge and a concave corner of a neighboring shape.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum required spacing between the end-of-line edge and the concave corner.

Parameters

'width <i>f_width</i>	The constraint applies only if the width of the end-of-line edge is less than or equal to this value.
'minLength <i>f_minLength</i>	The constraint applies only if both adjoining edges of an end-of-line edge are greater than or equal to this value.
'dualAdjacentLength (<i>f_minAdjLength1</i> <i>f_minAdjLength2</i>)	(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the edges forming the concave corner satisfy the following conditions: ■ Any one edge is greater than the first value ■ The other edge is greater than the second value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Examples

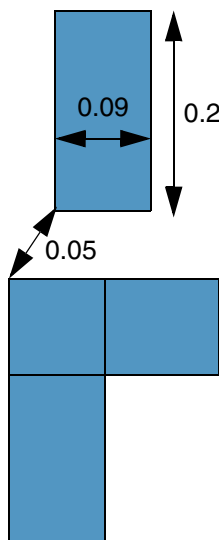
- Example 1: minEndOfLineToConcaveCornerSpacing with width and minLength
- Example 2: minEndOfLineToConcaveCornerSpacing with width and dualAdjacentLength

Example 1: minEndOfLineToConcaveCornerSpacing with width and minLength

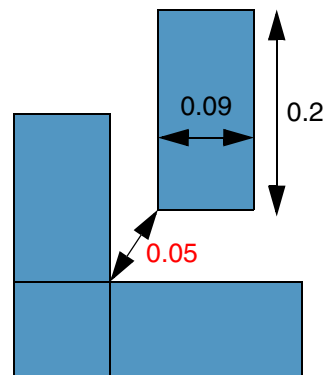
The minimum spacing between an end-of-line edge less than or equal to 0.1 wide and a concave corner must be greater than or equal to 0.06 if both adjoining edges of the end-of-line edge are greater than or equal to 0.2 long.

```
spacings (  
  ( minEndOfLineToConcaveCornerSpacing "Metal1"  
    'width 0.1  
    'minLength 0.2  
    0.052  
  )  
) ;spacings
```

■ Metal1



a) The constraint does not apply because the corner is not a concave corner.



b) FAIL. The width of the end-of-line edge is 0.09 (<0.1) and the length of both its adjoining edges is 0.2, but the spacing between the end-of-line edge and the concave corner is 0.05 (<0.052).

Virtuoso Technology Data Constraints Reference

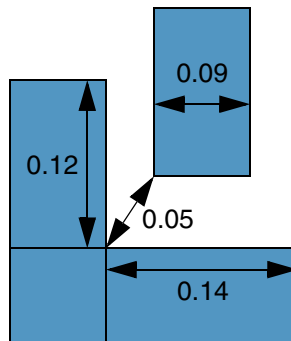
Spacing Constraints (One Layer)

Example 2: minEndOfLineToConcaveCornerSpacing with width and dualAdjacentLength

The minimum spacing between an end-of-line edge less than or equal to 0.1 wide and a concave corner must be greater than or equal to 0.06 if the edges that form the concave corner are both greater than 0.12 long.

```
spacings (  
  ( minEndOfLineToConcaveCornerSpacing "Metal1"  
    'width 0.1  
    'dualAdjacentLength (0.12 0.12)  
    0.06  
  )  
) ;spacings
```

■ Metal1



The constraint does not apply because the vertical edge (=0.12) forming the concave corner is not greater than 0.12. It would fail if 'dualAdjacentLength' was not specified because the spacing between the end-of-line edge and the concave corner is only 0.05 (<0.06).

minEndOfLineToNotchSpacing

```
spacings (
    ( minEndOfLineToNotchSpacing tx_layer
      'width f_width 'notchLength f_notchLength
      f_spacing
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between an end-of-line edge and a notch whose length is less than the specified notch length.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The required minimum spacing.

Parameters

'width <i>f_width</i>	The constraint applies only if the width of an end-of-line edge is less than this value.
'notchLength <i>f_notchLength</i>	The constraint applies only if the length of the notch is less than this value.

Virtuoso Technology Data Constraints Reference

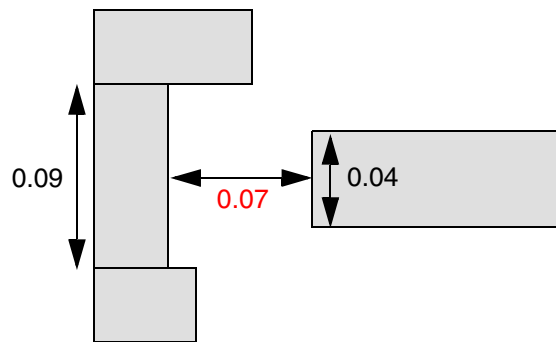
Spacing Constraints (One Layer)

Example

The distance between an end-of-line edge that is less than 0.05 wide and a notch that is less than 0.1 in length must be at least 0.08.

```
spacings(  
  ( minEndOfLineToNotchSpacing "Metal2"  
    'width 0.05 'notchLength 0.1  
    0.08  
  )  
) ;spacings
```

□ Metal2



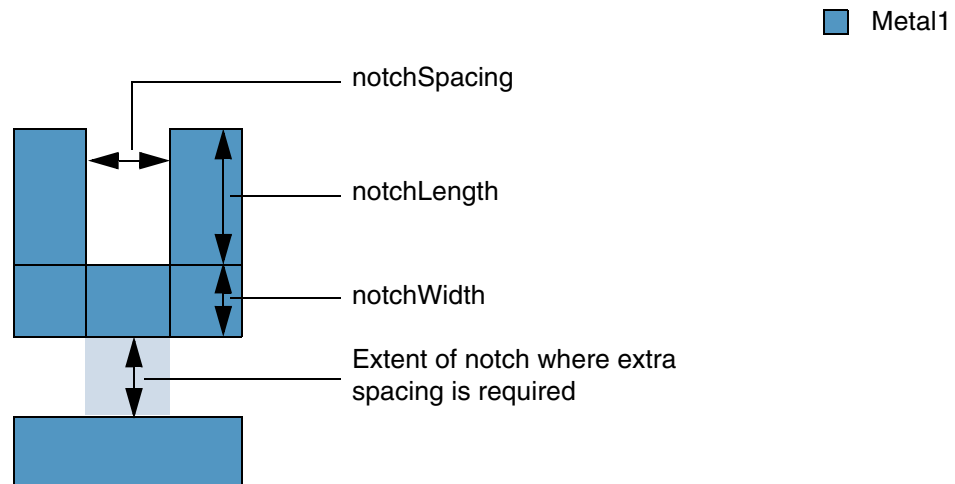
FAIL. The end-of-line width is 0.04 (<0.05) and the length of the notch is 0.09 (<0.1), but the distance of the end-of-line edge from the notch is 0.07 (<0.08).

minEndOfNotchSpacing

```
spacings (
    ( minEndOfNotchSpacing tx_layer
      'notchWidth f_notchWidth
      'notchLength f_notchLength
      'notchSpacing f_notchSpacing
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between the closed side of a U-shaped notch and any other shape on the layer with a parallel overlap with the notch.

The extent of the notch where extra spacing is required is shown below.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the notch and the neighboring shape must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

'notchWidth *f_notchWidth*

The constraint applies only if notch width is less than this value.

'notchLength *f_notchLength*

The constraint applies only if notch length is greater than or equal to this value.

'notchSpacing *f_notchSpacing*

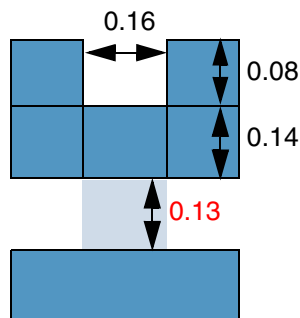
The constraint applies only if notch spacing is less than or equal to this value.

Example

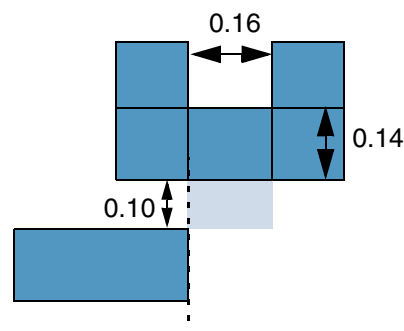
The spacing between the closed end of a notch and a shape that has a parallel overlap with the notch must be at least 0.14 if all of the following conditions are satisfied: notch width is less than 0.15, notch length is greater than or equal to 0.08, and notch spacing is less than or equal to 0.16.

```
spacings (  
  ( minEndOfNotchSpacing "Metal1"  
    'notchWidth 0.15  
    'notchLength 0.08  
    'notchSpacing 0.16  
    0.14  
  )  
) ;spacings
```

■ Metal1



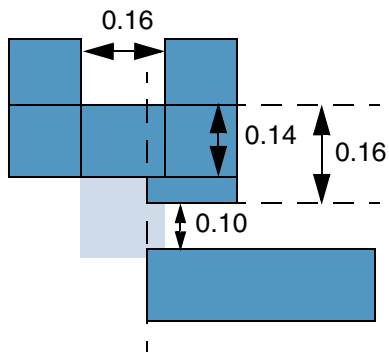
a) FAIL. Notch width is 0.14 (<0.15), notch length is 0.08, and notch spacing is 0.16, but the spacing between the two shapes is only 0.13 (<0.14).



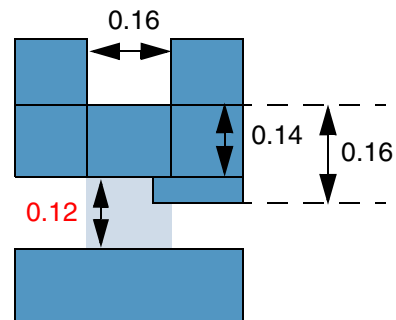
b) The constraint does not apply because there is no overlap with the notch.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply because the notch width at the point of overlap is 0.16 (>0.15).



d) FAIL. The notch width for part of the overlap is 0.14 (<0.15), but the spacing between the two shapes is only 0.12 (<0.14).

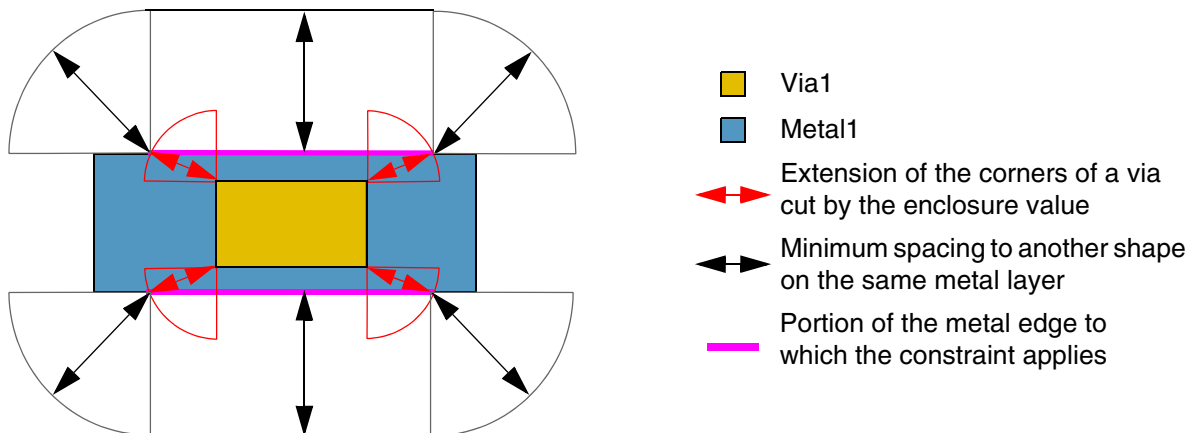
minExtensionSpacing

```
spacingTables (
  ( minExtensionSpacing tx_layer
    (( "extension" nil nil )
      ['cutClass {f_width | (f_width f_length) | t_name} ['longEdgeOnly]]
      ['layer tx_cutLayer]
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum spacing on a metal layer between an edge that has an extension less than the specified value past a shape on a cut layer to a second shape on the same metal layer.

Optionally, the constraint applies only to the metal edge containing the long edge of the specified rectangular cut class. The constraint can also be applied to a cut shape on the specified cut layer.

The portion of the metal edge to which the constraint applies is determined by extending the corners of the cut shape by the extension value in Euclidean style to intersect the metal edge.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"extension" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of the table row is as follows: (<i>f_extension f_spacing</i>) where, an edge with extension less than <i>extension</i> must have spacing greater than or equal to <i>spacing</i> . The corners of the cut shape are extended by a value equal to <i>extension</i> to determine the portion of metal edge to which the constraint applies.

Parameters

'cutClass { <i>f_width</i> (<i>f_width f_length</i>) <i>t_name</i> }	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class
'longEdgeOnly	The constraint applies only to the long edges of a rectangular via cut.
'layer <i>tx_cutLayer</i>	The constraint applies only to the via cuts on this cut layer.
<i>f_default</i>	The spacing value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

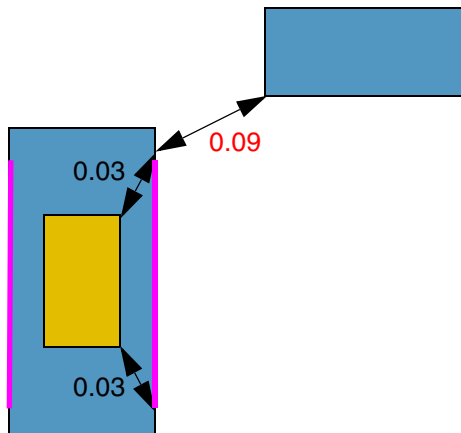
Spacing Constraints (One Layer)

Example

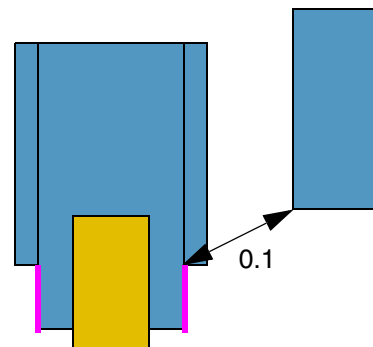
The spacing between a Metal1 edge that has an extension less than 0.03 past a cut shape of type VB (0.04x0.06) on layer Via1 to another Metal1 shape must be at least 0.1.

```
spacingTables(  
  ( minExtensionSpacing "Metal1"  
    (( "extension" nil nil )  
      'cutClass "VB" 'longEdgeOnly  
      'layer "Via1"  
    )  
    (0.03 0.1)  
  )  
);spacingTables
```

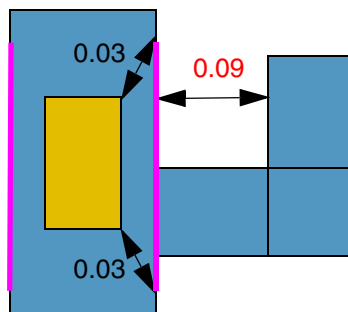
■ Via1
■ Metal1



a) FAIL. The corners of the via cut are extended by 0.03 to determine the metal edges (in pink) that must satisfy the spacing requirement of 0.1, but the spacing is only 0.09 (<0.1).



b) PASS. The spacing of 0.1 applies only to the portion of the metal edge (in pink) containing the cut shape with enclosure less than 0.03.

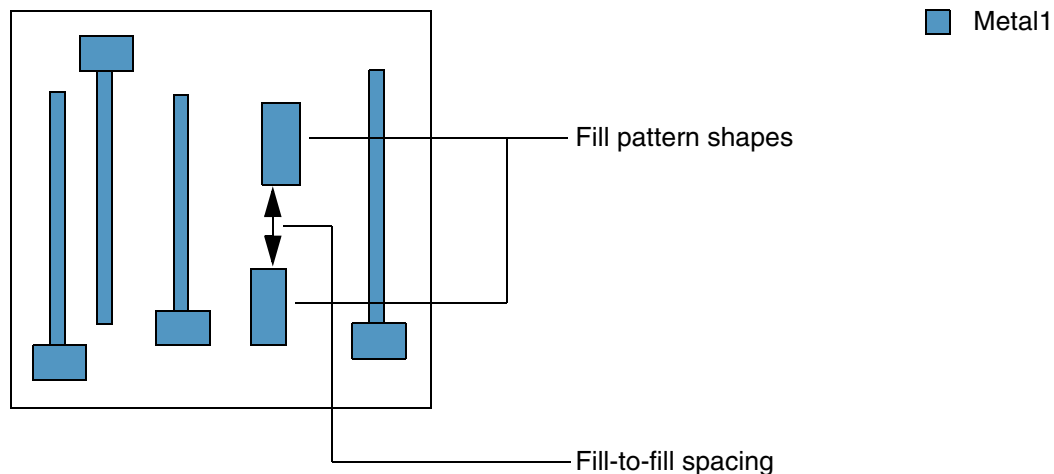


c) FAIL. The spacing between the portion of the metal edge to which the constraint applies (in pink) and another edge of the contiguous same-metal shape must be 0.1, but the spacing is only 0.09 (<0.1).

minFillToFillSpacing

```
spacings (
    ( minFillToFillSpacing tx_Layer
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between the fill shapes on a layer.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between fill shapes must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

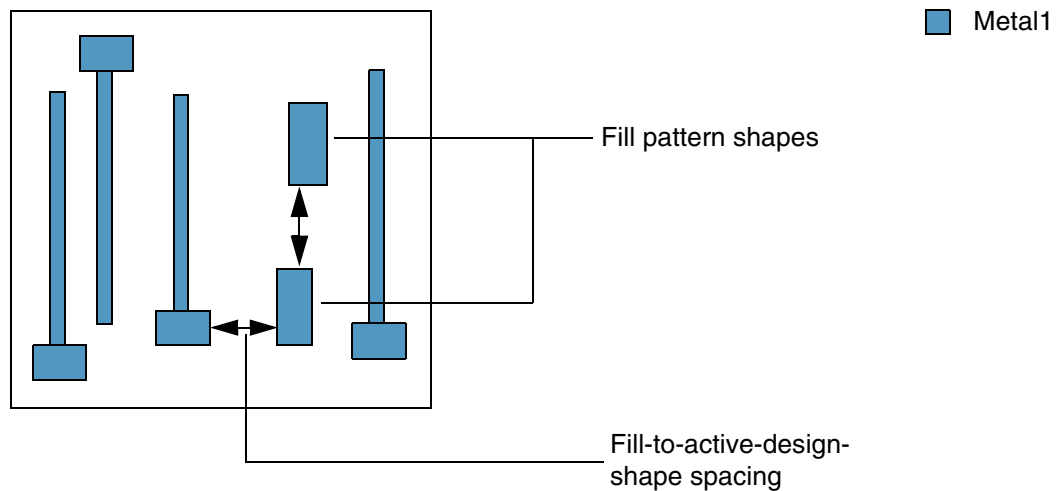
The minimum spacing between fill shapes on Metal1 must be at least 0.2.

```
spacings (
  ( minFillToFillSpacing "Metal1"
    0.2
  )
) ;spacings
```

minFillToShapeSpacing

```
spacings (
    ( minFillToShapeSpacing tx_layer
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between fill shapes and active design shapes on a layer.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between a fill shape and an active design shape must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The minimum spacing between fill shapes and active design shapes must be greater than or equal to 1.0 on Metal1 and greater than or equal to the value of the technology parameter minfillt2objspace1 on Metal2.

```
spacings (
  ( minFillToShapeSpacing "Metal1"
    1.0
  )
  ( minFillToShapeSpacing "Metal2"
    techParam("minfillt2objspace1")
  )
) ;spacings
```

minFiveWiresEndOfLineSpacing

```
orderedSpacings (
  ( minFiveWiresEndOfLineSpacing tx_metalLayer tx_cutLayer
    'width f_width
    'parallelEdgeSpace f_paraEdgeSpacing
    'distance f_eolWithin
    'enclosedDistance f_enclosedDistance
    'cutDistance f_cutWithin
    'prl f_prl
    'noMetalEolExtension (f_sideExt f_forwardExt)
    f_eolSpacing
  )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Only) Specifies that a wire must not overlap with the search region formed by *eolWithin*—measured along the end-of-line edge of the middle wire of a group of five wires—and *eolSpacing*—measured orthogonal to the end-of-line edge—provided the following conditions are true:

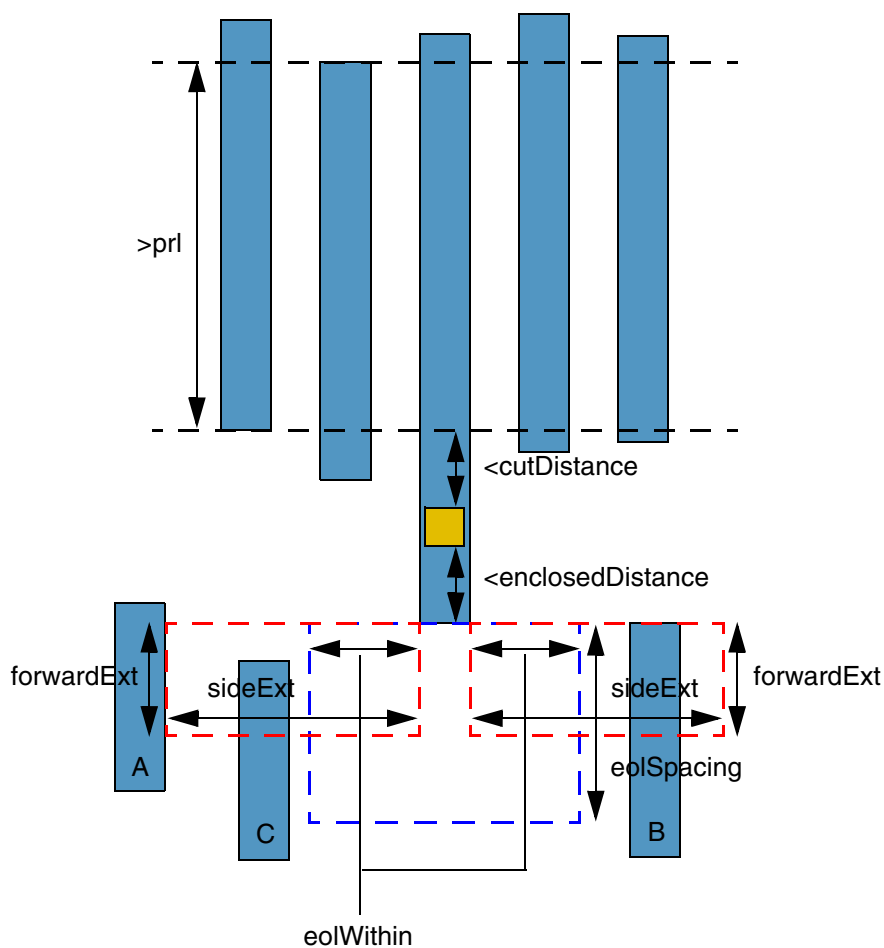
- Each wire in the group of five wires has width equal to *width* and each wire in the group is spaced *paraEdgeSpacing* apart, with common parallel run length greater than *prl*.
- The via cut enclosed by the middle wire is at a distance less than *cutWithin* from the group of five wires, the distance measured beyond the common parallel run length.
- The distance of the via cut from the end-of-line edge of the middle wire is less than *enclosedDistance*.
- No wire completely overlaps the two search regions, including touching, formed by *sideExt* (measured along the end-of-line edge) and *forwardExt* (measured orthogonal to the end-of-line edge), in the orthogonal direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The width of each wire in this set of five wires is equal to *width* and the distance of each wire from its neighboring wires is equal to *paraEdgeSpacing*.

Via1
 Metal1



A violation occurs if a wire overlaps with the blue region when a wire like A or B (the wire labeled C is irrelevant) covering the entire y range of the red regions does not exist.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_eolSpacing</i>	The minimum spacing required between the end-of-line edges of the middle wire containing the via and a neighboring wire.

Parameters

'width <i>f_width</i>	The constraint applies only if the width of each of the five wires is equal to this value.
'parallelEdgeSpace <i>f_paraEdgeSpacing</i>	The constraint applies only if the spacing between each pair of adjacent wires in the set of five wires is equal to this value.
'distance <i>f_eolWithin</i>	The end-of-line edge of the middle wire containing the via cut is extended by this value on both sides when searching for a neighboring wire beyond the end-of-line edge. This parameter together with the constraint value (<i>eolSpacing</i>) forms a search window. A violation occurs if a wire overlaps this search window when all required conditions are met.
'enclosedDistance <i>f_enclosedDistance</i>	The constraint applies only if the via cut is at a distance less than this value from the end-of-line edge of the middle wire.
'cutDistance <i>f_cutDistance</i>	The constraint applies only if the via cut is at a distance less than this value from the group of five wires with common parallel run length greater than <i>prl</i> .

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'prl f_prl`

The constraint applies only if the five wires have common parallel run length greater than this value.

`'noMetalEolExtension (f_sideExt f_forwardExt)`

The two values, *sideExt* (measured along the end-of-line edge) and *forwardExt* (measured orthogonal to the end-of-line edge), together define two additional search regions on either side of the middle wire. The constraint applies only if no wire completely overlaps any of these search regions, including touching, in the orthogonal direction.

Example

A neighboring wire must not overlap a search region that extends 0.02 on either side of the end-of-line edge of the middle wire (of a set of five wires) and 0.1 in the direction orthogonal to it if all of the following conditions are true:

- The width of each wire in the set is 0.04 and the wires are spaced 0.04 apart, with common parallel run length greater than 0.14.
- The middle wire contains a via cut that is at a distance less than 0.03 from the end-of-line edge of the wire and at a distance less than 0.13 measured beyond the common parallel run length of the group of wires.
- No wire overlaps the search regions formed by the value pair (0.15, 0.08).

```
orderedSpacings (
  ( minFiveWiresEndOfLineSpacing "Metal3" "Via2"
    'width 0.04
    'parallelEdgeSpace 0.04
    'distance 0.02
    'enclosedDistance 0.03
    'cutDistance 0.13
    'prl 0.14
    'noMetalEolExtension (0.15 0.08)
    0.1
  )
) ;orderedSpacings
```


minHoleWidth

```

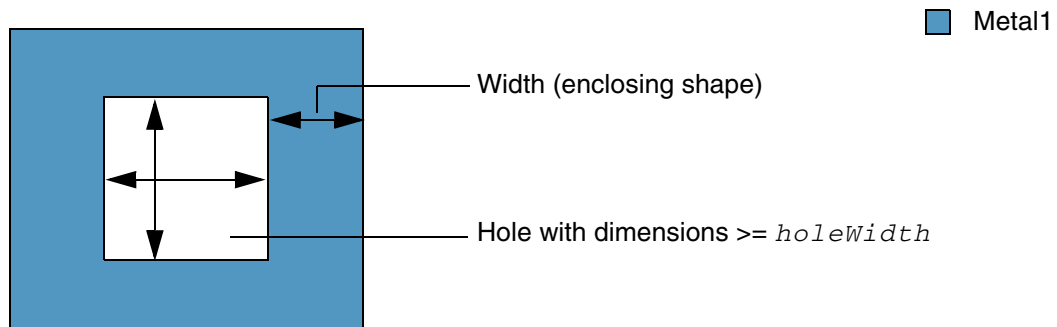
spacings (
    ( minHoleWidth tx_layer
      f_holeWidth
    )
) ;spacings

spacingTables (
    ( minHoleWidth tx_layer
      (( "width" nil nil )
        [f_default]
      )
      (g_table)
    )
) ;spacingTables

```

Specifies the minimum dimensions of a hole (that is, the spacing between two opposite internal edges of a hole) in a donut shape on a layer, in both X and Y directions.

This constraint is used in conjunction with the [minHoleArea](#) constraint because it is possible for geometries to meet the minimum area requirement, but not the minimum spacing requirement specified using `minHoleWidth`.



Values

`tx_layer`

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

`f_holeWidth`

The width of the hole in both X and Y directions must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

"width" nil nil

This identifies the index for *table*.

g_table

The format of the *table* row is as follows:

(*f_width f_value*)

where,

- *f_width* is the width of the donut shape.
- *f_value* is the minimum dimensions of the hole in both X and Y directions when the width of the donut shape is greater than or equal to the corresponding index.

Type: A 1-D table specifying floating point width values

Parameters

f_default

The hole width value to be used when no table entry applies.

Example

The area of a hole on Metal1 must be greater than or equal to 1.5 and the width of the hole in both X and Y directions must be greater than or equal to 1.0.

```
spacings (
  ( minHoleArea "Metal1"
    1.5
  )
  ( minHoleWidth "Metal1"
    1.0
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

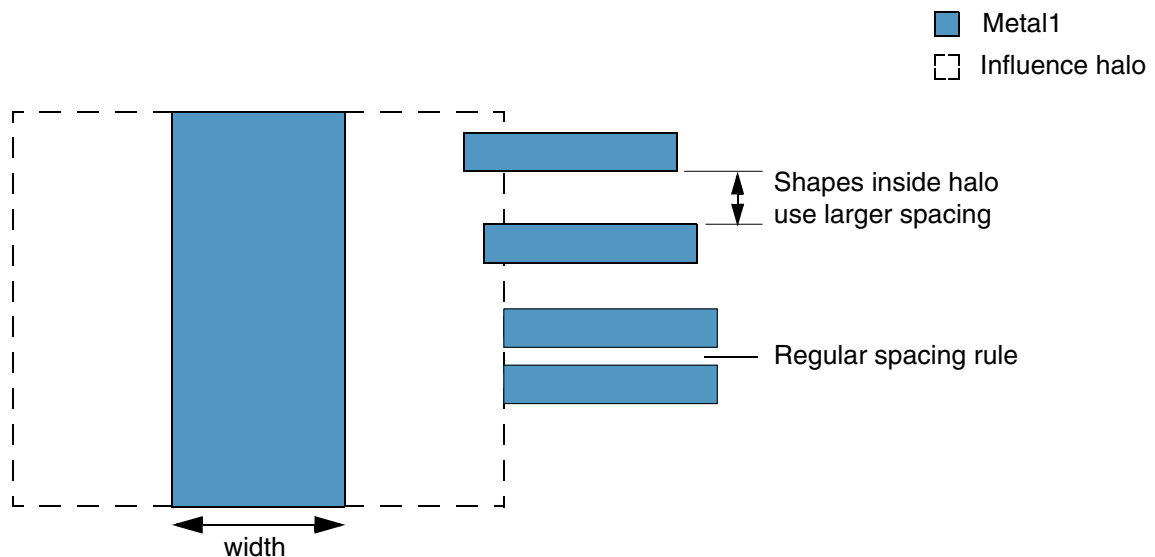
minInfluenceSpacing

```
spacings (
  ( minInfluenceSpacing tx_layer
    f_spacing
  )
) ;spacings

spacingTables (
  ( minInfluenceSpacing tx_layer
    (( "width" nil nil ["distance" nil nil] )
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum spacing between two shapes on a layer if they are within a certain distance of a large shape.

Also known as the proximity or influence rule, the spacing required by smaller shapes is influenced by the large shapes nearby, that is, the smaller shapes pick up the spacing required by a wider neighboring shape.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between shapes inside the influence halo must be greater than or equal to this value.
"width" nil nil ["distance" nil nil]	This identifies the index for <i>table</i>.
<i>g_table</i>	The format of a 1-D table is as follows: <i>(f_width f_spacing)</i> where, <i>f_width</i> is the width of the large influencing shape and <i>f_spacing</i> is the minimum spacing required between the shapes when the width of the large influencing shape is greater than or equal to the corresponding index value. The format of a 2-D table is as follows: <i>(f_width f_distance) f_spacing</i> where, ■ <i>f_width</i> is the width of the large influencing shape and <i>f_distance</i> is the distance up to which the influence halo extends from the large influencing shape. ■ <i>f_spacing</i> is the minimum spacing required between shapes inside the influence halo when both width and distance are greater than or equal to the corresponding index values. Type: A 1-D table specifying floating-point width and spacing values, or a 2-D table specifying floating-point width, distance, and spacing values

Parameters

<i>f_default</i>	The spacing value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The minimum spacing between shapes inside the influence halo of a Poly1 shape must be at least 0.5, and the minimum spacing between shapes inside the influence halo of a Metal2 shape must be greater than or equal to the value of the technology parameter `minspacing1`.

```
spacings (
  ( minInfluenceSpacing "Poly1"
    0.5
  )
  ( minInfluenceSpacing "Metal2"
    techParam("minspacing1")
  )
) ;spacings
```

minJointCornerSpacing

```
spacings (
    ( minJointCornerSpacing tx_layer
      'jointWidth f_width
      'spanLength f_spanLength
      ['sameMask]
      ['length f_length]
      ['minLength f_minLength]
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing on a layer between two corners formed by joint edges when the parallel run length between the corners is less than or equal to zero.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between corners must be greater than or equal to this value.

Parameters

<i>'jointWidth f_width</i>	The constraint applies only if the width of the joint (that is not an end-of-line edge) is less than this value.
<i>'spanLength f_spanLength</i>	The constraint applies only if the span length of the joint is greater than this value.
<i>'sameMask</i>	(Virtuoso Advanced Node for Layout Standard) The constraint applies only between shapes on the same mask. Type: Boolean
<i>'length f_length</i>	The constraint applies only if the length of the edges forming the corners is less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'minLength f_minLength`

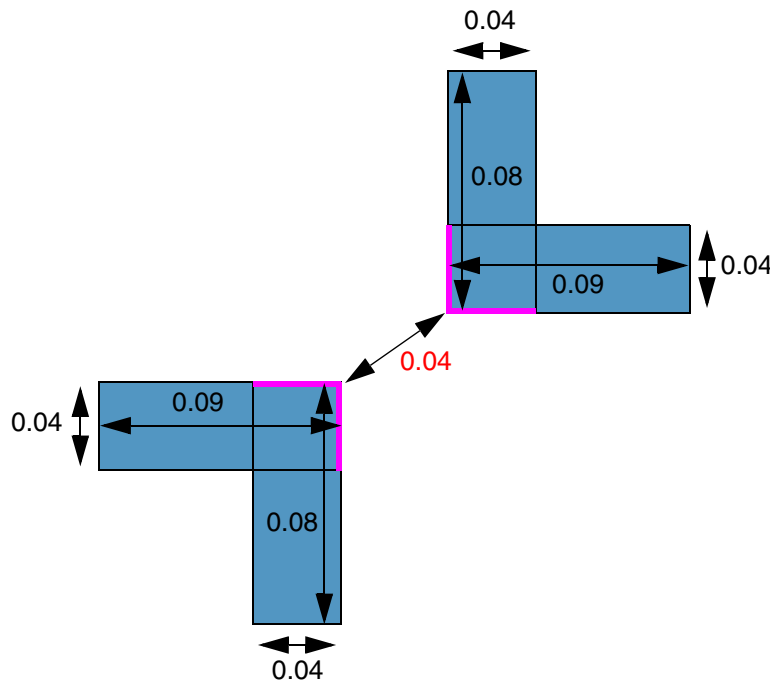
The constraint applies only if the length of both sides of a joint (that is not an end-of-line edge) is greater than or equal to this value.

Example

The spacing between two corners formed by joint edges must be at least 0.05 if the joint width is less than 0.07 and the joint span length is greater than 0.06. Additionally, the length of the edges forming the corners must be less than 0.1.

```
spacings (  
  ( minJointCornerSpacing "Metal1"  
    'jointWidth 0.07  
    'spanLength 0.06  
    'length 0.1  
    0.05  
  )  
) ;spacings
```

■ Metal1



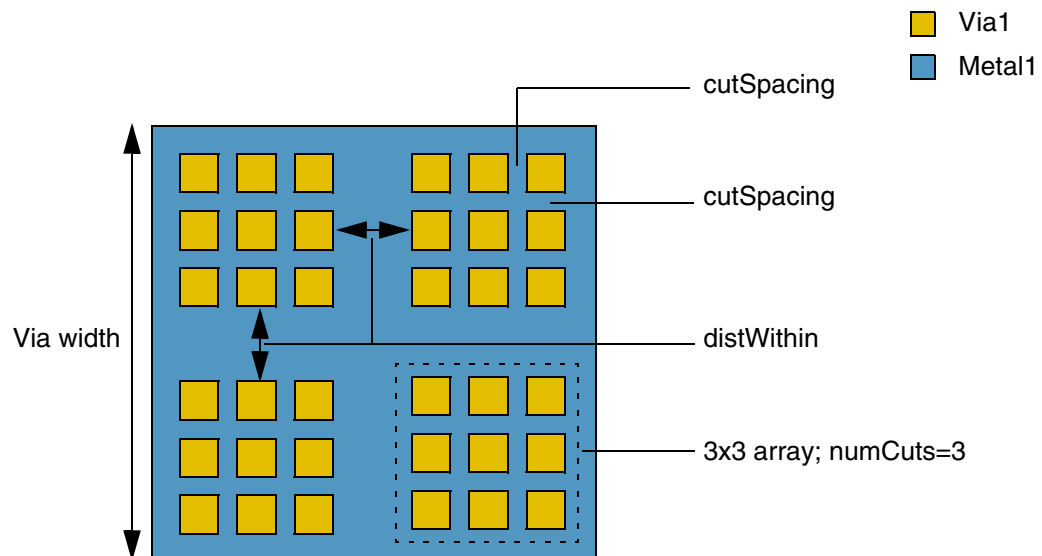
FAIL. The joint width is 0.04 (<0.07), the span lengths are 0.08 and 0.09 (both greater than 0.06), and the lengths of the edges that form the corners are both less than 0.1. However, the spacing between the corners is only 0.04 (<0.05).

minLargeNeighborViaArrayCutSpacing

```
spacings (
    ( minLargeNeighborViaArrayCutSpacing tx_cutLayer
      'numCuts x_numCuts
      ['cutClass {f_width | (f_width f_length) | t_name}]
      ['distanceWithin f_distWithin 'arrayWidth f_arrayWidth]
      f_cutSpacing
    )
) ;spacings
```

Specifies the minimum spacing between via cuts in a large array with rows and columns greater than or equal to *numCuts*.

Optionally, the constraint can be applied to the via cuts in a neighboring array if the neighboring array is at a distance less than *distWithin* from the large array and has parallel run length greater than or equal to *arrayWidth*, which is measured in the direction perpendicular to *distWithin*.



Values

tx_cutLayer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_cutSpacing

The spacing between via cuts must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'numCuts x_numCuts`

The constraint applies only if the number of rows and columns in the large array is greater than or equal to this value.

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'distanceWithin f_distWithin 'arrayWidth f_arrayWidth`

The constraint applies to via cuts in a neighboring array if it is at a distance less than `distWithin` from the large array and has parallel run length greater than or equal to `arrayWidth`, which is measured in the direction perpendicular to `distWithin` (the via cuts in such a neighboring array are considered part of the large array).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

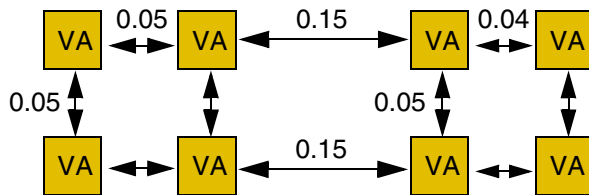
The spacing between VA via cuts in a 3x3 array and in a neighboring array at a distance less than 0.2 from the large array with width greater than or equal to 0.06 must be at least 0.05.

```

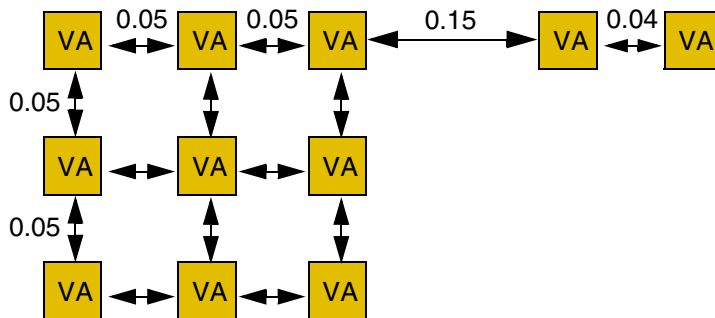
spacings (
    ( minLargeNeighborViaArrayCutSpacing "Via1"
      'numCuts 4
      'cutClass "VA"
      'distanceWithin 0.2 'arrayWidth 0.06
      0.05
    )
) ;spacings

```

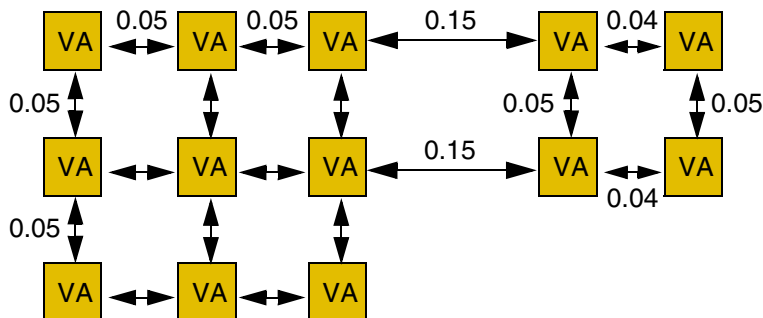
■ Via1



a) The constraint does not apply because the size of the large array is less than 3x3.



b) PASS. The via cuts in the large array on the left have a spacing of 0.05. The constraint does not apply to the neighboring array because the bounding box of the two via cuts is less than 0.06 wide.



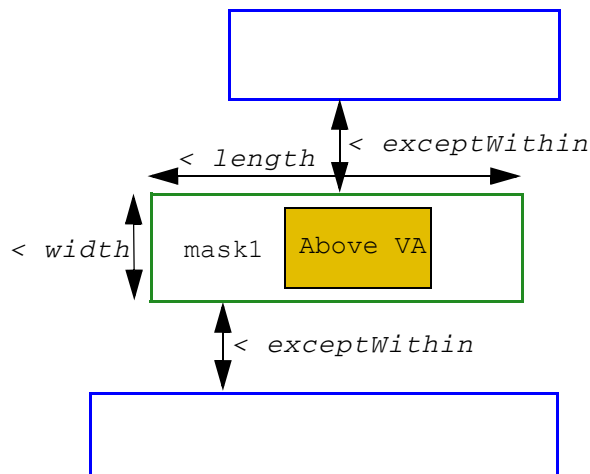
c) FAIL. The neighboring array on the right is at a distance less than 0.2 from the large array and has width greater than 0.06. However, the via cuts in this array have a spacing of only 0.04 (<0.05).

minLengthWithCut

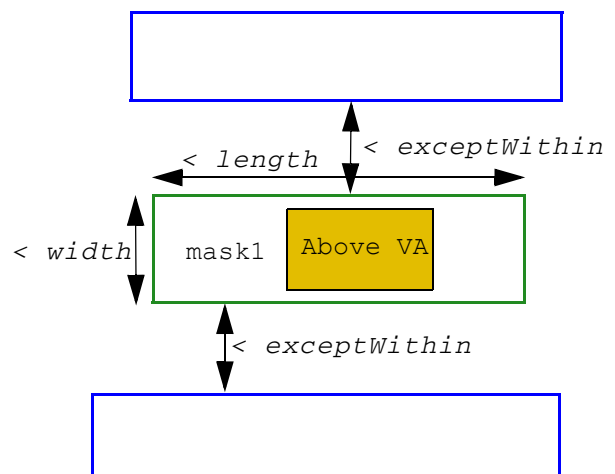
```
orderedSpacings (
  ( minLengthWithCut tx_metal tx_cut
    'width f_width
    {'mask1 | 'mask2 | 'mask3}
    'exceptWithin f_exceptWithin
    'eolSpacing f_eolSpacing
    ['cutClass f_width | (f_width f_length) | t_name]
    (f_length)
  )
); orderedSpacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the minimum length requirement on a wire with a cut and without neighbor wires covering the entire wire on two opposite sides within a specified distance.

Consider the following illustrations, where the constraint specifies the mask as mask1 and the cutClass as VA.



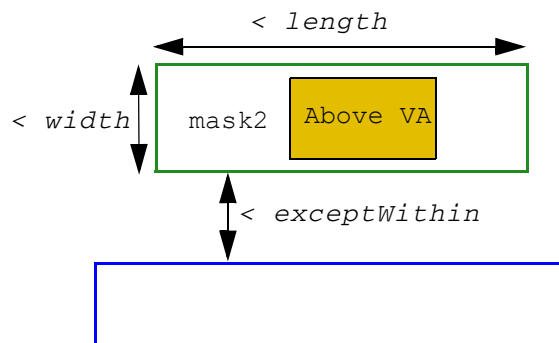
This is a violation because the top neighbor wire does not completely cover the middle wire,



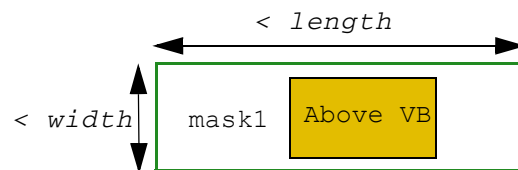
The constraint passes because the neighbor wires on both sides are within the exceptWithin distance.

Virtuoso Technology Data Constraints Reference

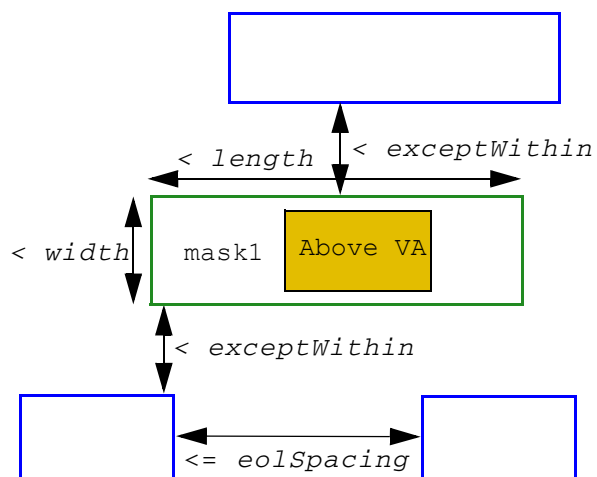
Spacing Constraints (One Layer)



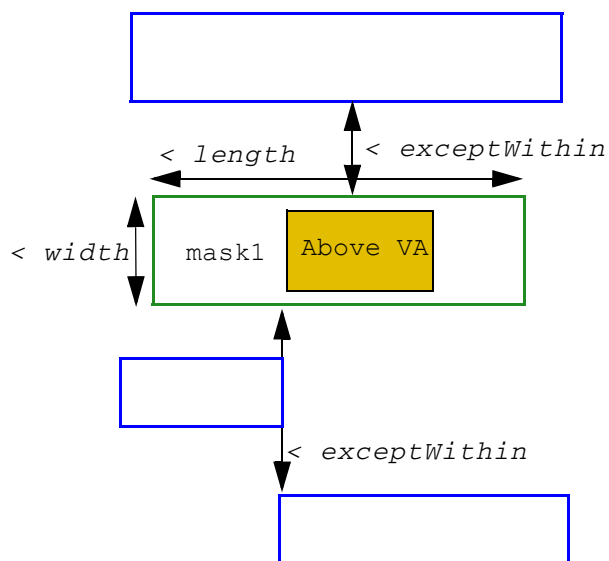
The constraint passes because the mask is not mask1.



The constraint passes because the cutClass is not VA.



The constraint passes. Because the two bottom wires have spacing $\leq$ eolSpacing, the gap could be considered to be filled in to completely cover the trigger wire at the bottom.



The constraint passes. The two bottom wires being within the exceptWithin distance are combined to completely cover the trigger wire at the bottom.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_metal</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cut</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_length</i>	The minimum length of the wire.

Parameters

<i>'width f_width</i>	The constraint applies only when the wire width is less than this value.
<i>'mask1 'mask2 'mask3</i>	The constraint applies only when the wire is on the specified mask.
<i>'cutClass {f_width (f_width f_length) t_name}</i>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class. If not specified, any cut combination of four cuts can trigger the rule.
<i>'exceptWithin f_exceptWithin</i>	The constraint exempts a wire if neighbor wires cover the entire wire within the given value on two opposite sides. If the given value is large enough to include multiple wires on one side, the constraint considers the combined projection to the triggering wire.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'eolSpacing f_eolSpacing`

If the spacing of the neighbor wire to another wire is less than or equal to the given value with parallel run length being more than 0, the gap is virtually filled in to include the neighbor.

Example

Specifies the minimum length of a wire with a VA cut with width 0.1 on mask1 within a distance of 0.07.

```
orderedSpacings(  
  (minLengthWithCut "Metal1" "Via1"  
    'width 0.1  
    'mask1  
    'exceptWithin 0.05  
    'eolSpacing 0.07  
    'cutClass "VA"  
    0.07  
  )  
);orderedSpacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minNeighborVoltageSpacing (One layer)

```
spacings (
  ( minNeighborVoltageSpacing tx_layer
    ['horizontal | 'vertical]
    'otherLayer tx_otherLayer
    'otherDistance f_otherDistance
    'deltaVoltage f_voltage
    f_spacing
  )
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the spacing based on a neighboring shape that meets the delta voltage condition.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shapes must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'horizontal | 'vertical`

The direction in which the spacing is measured. If the direction is not specified, spacing is measured in any direction.

`'otherLayer tx_otherLayer`

The layer used for `f_otherDistance`.

`'otherDistance f_otherDistance`

The constraint only applies when there is a shape on `tx_otherLayer` within this distance from the shape on `tx_layer` on the same side where the spacing is measured.

`'deltaVoltage f_voltage`

The constraint applies if the delta voltage between the `layer1` shape and `layer2` shape is greater than this value.

The delta voltage is calculated as following:

- For asynchronous nets:

$\max(V1_{\max}, V2_{\max}) - \min(V1_{\min}, V2_{\min})$

- For synchronous nets:

$\max(|V1_{\max} - V2_{\max}|, |V1_{\min} - V2_{\min}|)$

Examples

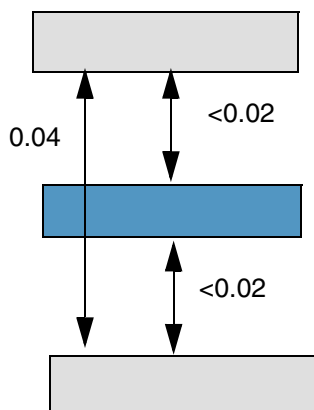
In this example, the maximum delta voltage is assumed to be > 1.1 . A minimum vertical spacing of 0.03 is required between shapes on layer `M1` if there is an `M2` shape within 0.02 of the `M1` shape on the same side as the `M1` shape.

Virtuoso Technology Data Constraints Reference

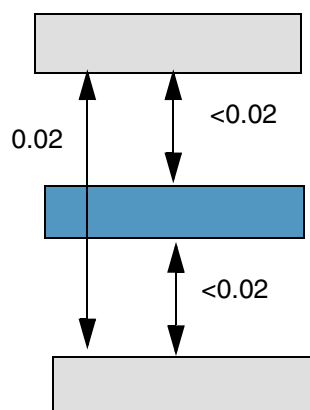
Spacing Constraints (One Layer)

```
spacings (  
  ( minNeighborVoltageSpacing "M1"  
    'vertical  
    'otherLayer "M2"  
    'otherDistance 0.02  
    'deltaVoltage 1.1  
    0.03  
  )  
) ;spacings
```

□ M1
■ M2



Pass: Spacing is 0.04, which is greater than the rule, ≥ 0.03



Fail: Spacing is 0.02, which is greater than the rule, ≥ 0.03

minNotchSpacing

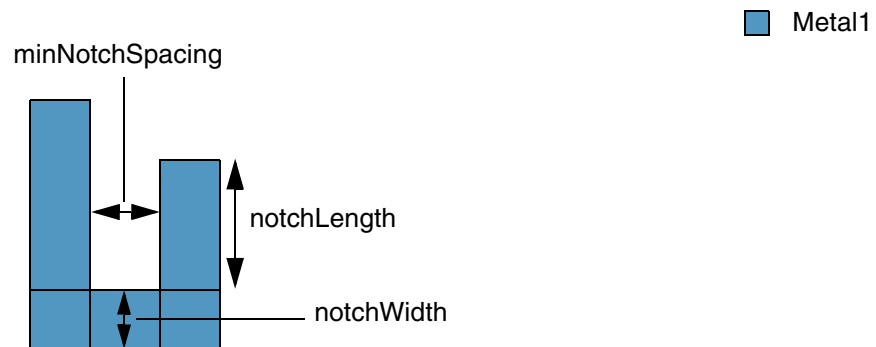
```

spacings (
    ( minNotchSpacing tx_layer
      ['horizontal | 'vertical]
      ['notchLength f_notchLength]
      [['within f_within 'maxSpanLength f_spanLength]
        | ['width f_width
          | 'concaveEndsWidth f_concaveEndsWidth
          | 'betweenConcaveCorners
        ]
      ]
      ['notchWidth f_notchWidth]
      ['excludeSpacing (g_range)]
      f_spacing
    )
) ;spacings

```

Specifies the minimum notch spacing for a layer. This spacing value applies if the length of the shorter side of the notch is less than the specified value.

Optionally, you can also specify a range of legal notch spacings that are exempted from the check. Other optional parameters specify when the constraint applies.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum notch spacing.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'horizontal | 'vertical`

(Virtuoso Advanced Node for Layout Only) If this parameter is not specified, the constraint applies to all notches. When `horizontal` is used, the constraint applies to a vertical notch only when there is horizontal spacing between the sides of the notch. When `vertical` is used, the constraint applies to a horizontal notch only when there is vertical spacing between the sides of the notch.

`'notchLength f_notchLength`

The constraint applies if the length of the shorter side of the notch is less than this value.

If `notchLength` is zero, the constraint always applies, regardless of the length of the notch.

`'within f_within 'maxSpanLength f_spanLength`

The constraint applies only if a portion of a side of the notch with span length less than `spanLength` and length greater than or equal to `notchLength` is at a distance less than `within` from the notch.

`'width f_width`

The constraint applies only if the width of at least one side of the notch is greater than or equal to this value.

`'concaveEndsWidth f_concaveEndsWidth`

The constraint applies only if the widths of both sides of the notch are less than or equal to this value. Additionally, the length of one of the sides must be less than `notchLength` and this side must be between two concave corners, and the length of the opposite side must be greater than or equal to `notchLength`.

`'betweenConcaveCorners`

The constraint applies only if the shorter of the two sides of the notch is between two concave corners.

Type: Boolean

`'notchWidth f_notchWidth`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the width of the notch is less than this value.

`'excludeSpacing (g_range)`

(Virtuoso Advanced Node for Layout Only) The constraint does not apply if the notch spacing falls in this range.

Type: Floating-point values specifying a range of spacing that is exempted

Examples

- Example 1: minNotchSpacing with notchLength
- Example 2: minNotchSpacing with notchLength and notchWidth
- Example 3: minNotchSpacing with notchLength and concaveEndsWidth
- Example 4: minNotchSpacing with notchLength, within, and maxSpanLength
- Example 5: minNotchSpacing with notchWidth and excludeSpacing

Virtuoso Technology Data Constraints Reference

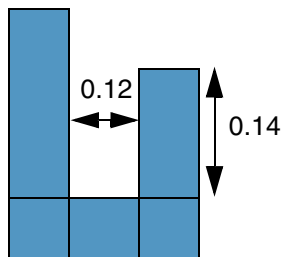
Spacing Constraints (One Layer)

Example 1: *minNotchSpacing* with *notchLength*

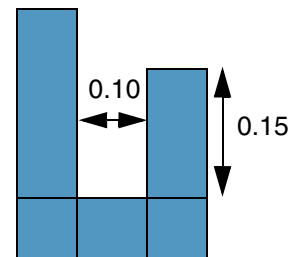
The notch spacing must be at least 0.12 if the shorter side of the notch is less than 0.15 long.

```
spacings (  
  ( minNotchSpacing "Metal1"  
    'notchLength 0.15  
    0.12  
  )  
) ;spacings
```

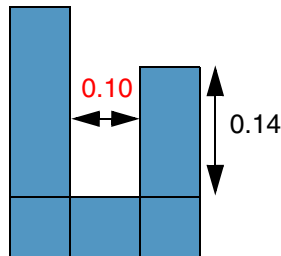
■ Metal1



a) PASS. The shorter side is 0.14 (<0.15) and the notch spacing is 0.12.



b) The constraint does not apply because the shorter side is 0.15; it must be less than 0.15 for the constraint to apply.



c) FAIL. The shorter side is 0.14 (<0.15), but the notch spacing is 0.10 (<0.12).

Virtuoso Technology Data Constraints Reference

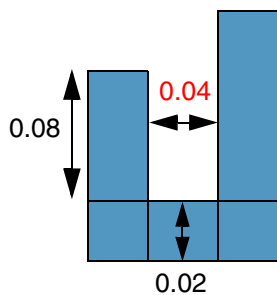
Spacing Constraints (One Layer)

Example 2: *minNotchSpacing* with *notchLength* and *notchWidth*

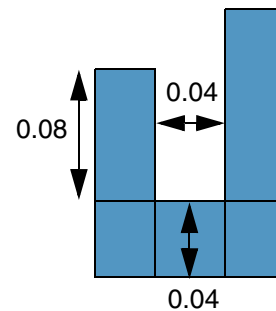
The notch spacing must be at least 0.05 if the shorter side of the notch is less than 0.09 long and the notch width is less than 0.03.

```
spacings (  
  ( minNotchSpacing "Metal1"  
    'notchLength 0.09  
    'notchWidth 0.03  
    0.05  
  )  
) ;spacings
```

■ Metal1



a) FAIL. The shorter side is 0.08 (<0.09) and the notch width is 0.02 (<0.03), but the notch spacing is 0.04 (<0.05).



b) The constraint does not apply because the notch width is 0.04 (>0.03).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

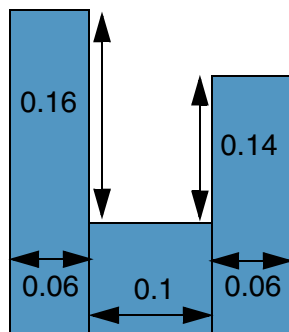
Example 3: *minNotchSpacing* with *notchLength* and *concaveEndsWidth*

The notch spacing must be at least 0.12 if the following conditions are met:

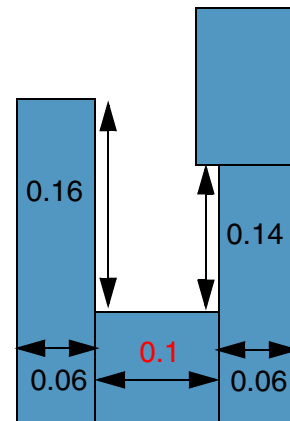
- Both sides of the notch are less than or equal to 0.08 wide.
- The shorter side is less than 0.15 long and lies between two concave corners, and the longer side is greater than or equal to 0.1.

```
spacings (  
  ( minNotchSpacing "Metal1"  
    'notchLength 0.15  
    'concaveEndsWidth 0.08  
    0.12  
  )  
) ;spacings
```

■ Metal1



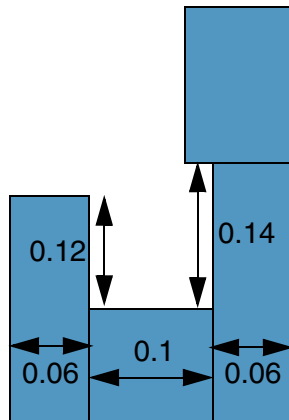
a) The constraint does not apply because the shorter side does not lie between two concave corners. A violation would occur if 'concaveEndsWidth was not specified.



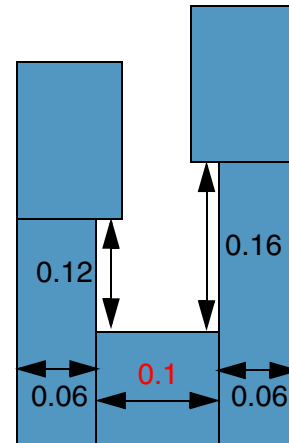
b) FAIL. Both sides of the notch are less than 0.08 wide; the shorter side is 0.14 (<0.15) and lies between two concave corners, and the longer side is 0.16 (>0.15). However, the notch spacing is 0.1 (<0.12).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply because the shorter side does not lie between two concave corners.



d) FAIL. Both sides of the notch are less than 0.08 wide; the shorter side is 0.12 (<0.15) and lies between two concave corners, and the longer side is 0.16 (>0.15). However, the notch spacing is 0.1 (<0.12).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

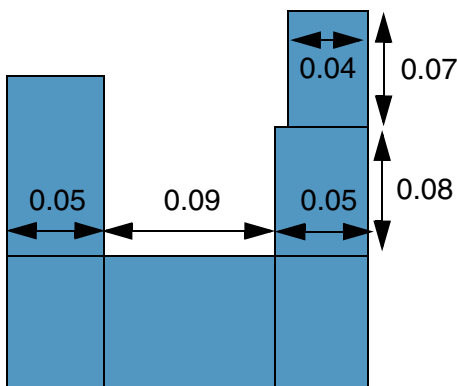
Example 4: *minNotchSpacing* with *notchLength*, *within*, and *maxSpanLength*

The notch spacing must be at least 0.1 if the following conditions are met:

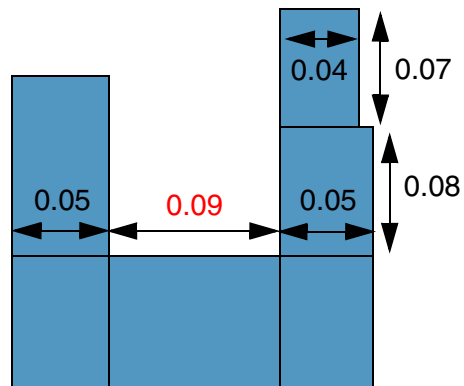
- The span length of at least a portion of the side that is within 0.2 of the notch is less than 0.05.
- The length of this portion is greater than or equal to 0.06.

```
spacings(  
  ( minNotchSpacing "Metal1"  
    'notchLength 0.06  
    'within 0.2 'maxSpanLength 0.05  
    0.1  
  )  
) ;spacings
```

■ Metal1



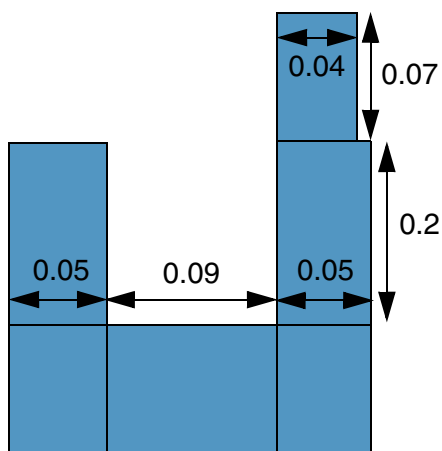
a) The constraint does not apply because the span length of the bottom portion of the right side of the notch (the portion that is considered) is equal to 0.05 (must be less than 0.05).



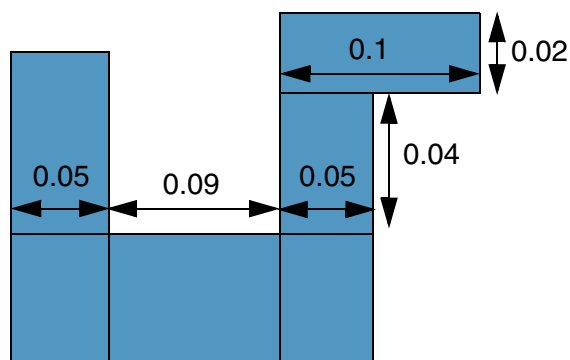
b) FAIL. The top portion of the right side of the notch is within 0.2 of the notch. Its span length is 0.04 (<0.05) and its length is 0.07 (>0.06), but the notch spacing is 0.09 (<0.1).

Virtuoso Technology Data Constraints Reference

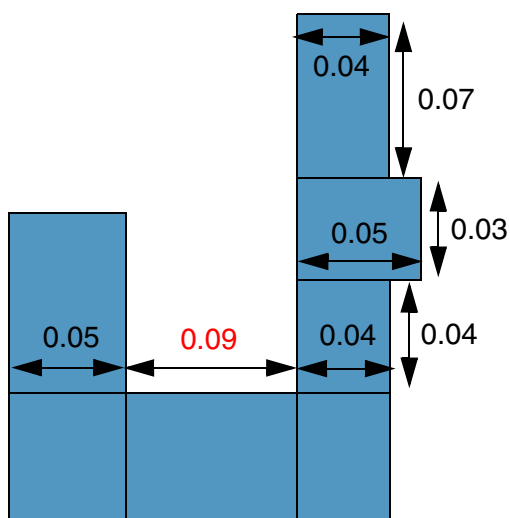
Spacing Constraints (One Layer)



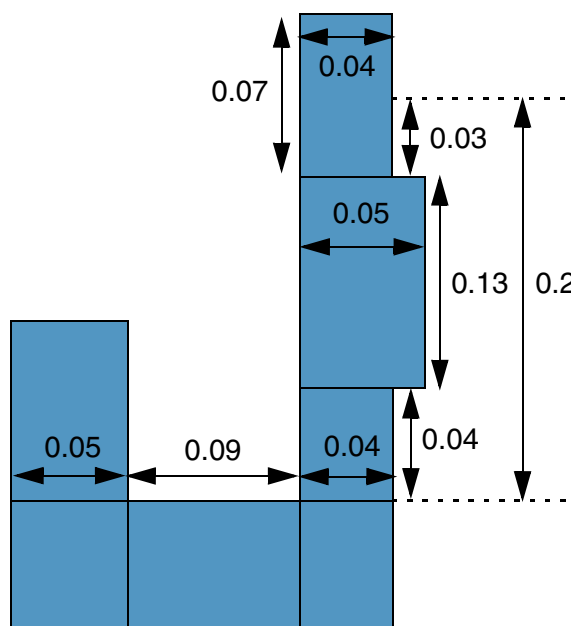
c) The constraint does not apply because the span length of the bottom portion of the right side of the notch (the portion that is within 0.2 of the notch) is equal to 0.05 (must be less than 0.05).



d) The constraint does not apply because the bottom portion of the right side of the notch has length 0.04 (<0.06) and the top portion has span length 0.1 (>0.05).



e) FAIL. The top portion of the right side of the notch is within 0.2 of the notch. Its span length is 0.04 (<0.05) and its length is 0.07 (>0.06), but the notch spacing is 0.09 (<0.1).



f) The constraint does not apply because the top portion of the right side of the notch, which has a span length of 0.04 (<0.05), has only a length of 0.03 (<0.06) within 0.2 of the notch.

Virtuoso Technology Data Constraints Reference

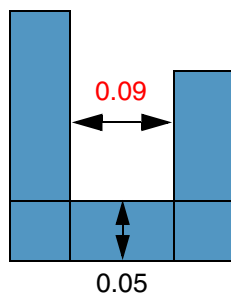
Spacing Constraints (One Layer)

Example 5: *minNotchSpacing* with *notchWidth* and *excludeSpacing*

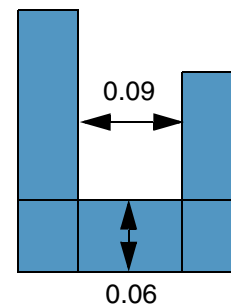
The notch spacing must be at least 0.1 if the notch width is less than or equal to 0.05. The constraint does not apply if the notch spacing is greater than or equal to 0.06 and less than 0.08.

```
spacings (  
  ( minNotchSpacing "Metal1"  
    'notchWidth 0.05  
    'excludeSpacing "[0.06 0.08)"  
    0.1  
  )  
) ;spacings
```

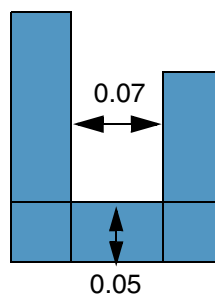
■ Metal1



a) FAIL. The notch width is 0.05, but the notch spacing is 0.09 (<0.1).



b) The constraint does not apply because the notch width is 0.06 (>0.05).



c) The constraint does not apply because the notch spacing of 0.07 is in the specified range.

minNotchSpanSpacing

```
spacings (
    ( minNotchSpanSpacing tx_layer
      'notchSpan f_notchSpan
      'notchSpace f_notchSpace
      ['exceptNotchLength f_notchLength]
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between a notch and a neighboring shape on a layer. The spacing applies only if the span of the legs of the notch is less than *notchSpan* and the distance between the legs of the notch is less than *notchSpace*. Additionally, the neighboring shape must have parallel run length greater than zero with both legs of the notch.

Multiple consecutive notches satisfying *notchSpan* and *notchSpace* constitute a violation, irrespective of the distance of the neighboring shape from a notch.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the notch and a neighboring shape must be greater than or equal to this value.

Parameters

'notchSpan <i>f_notchSpan</i>	The constraint applies only if the span of the legs of the notch is less than this value.
'notchSpace <i>f_notchSpace</i>	The constraint applies only if the distance between the legs of the notch is less than this value.
'exceptNotchLength <i>f_notchLength</i>	The constraint does not apply if the notch length is equal to this value.

Virtuoso Technology Data Constraints Reference

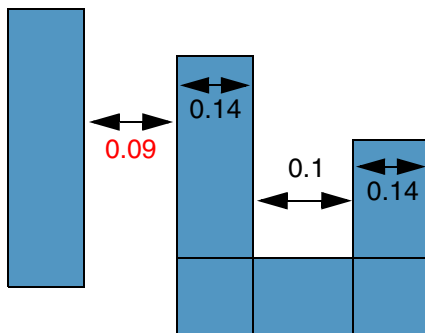
Spacing Constraints (One Layer)

Example

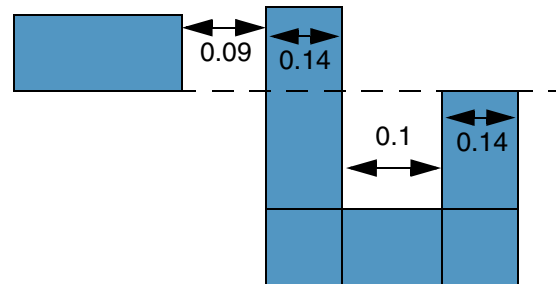
The spacing between a notch and a neighboring shape must be at least 0.1 if the span of the legs of the notch is less than 0.15 and the spacing between the legs of the notch is less than 0.12. Notches with length equal to 0.13 are exempt from the rule.

```
spacings (
  ( minNotchSpanSpacing "Metal1"
    'notchSpan 0.15
    'notchSpace 0.12
    'exceptNotchLength 0.13
    0.1
  )
) ;spacings
```

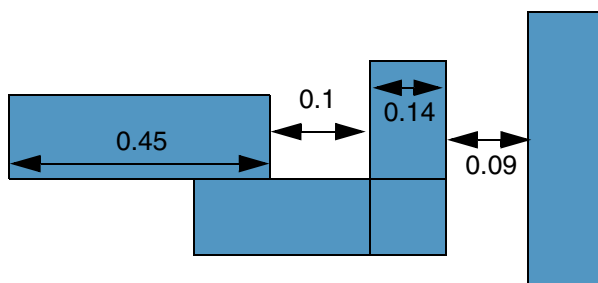
■ Metal1



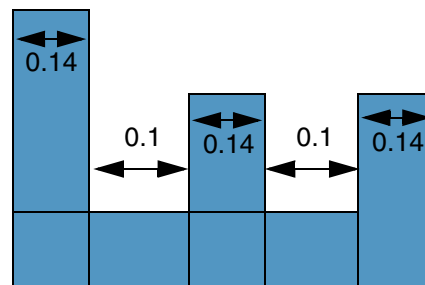
a) FAIL. The span of the legs of the notch is 0.14 (<0.15) and the distance between the legs of the notch is 0.1 (<0.12). However, the distance of the neighboring shape from the notch is only 0.09 (<0.1).



b) The constraint does not apply because the neighboring shape does not have a parallel run length greater than zero with both legs of the notch.



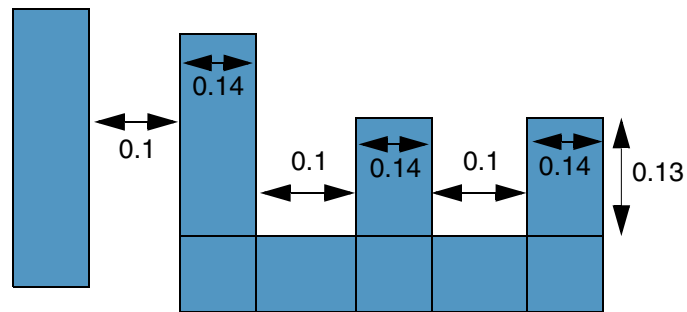
c) The constraint does not apply because the span of the left leg of the notch is 0.45 (>0.15).



d) FAIL. Multiple consecutive notches cause a violation, irrespective of the distance of the neighboring shape from a notch.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



e) The constraint does not apply because the length of a leg of the notch is equal to 0.13.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minOppSpanSpacing

```
spacings (
  ( minOppSpanSpacing tx_layer
    'width f_width 'eolWidth f_eolWidth ['length f_length]
    ['sameMask]
    ['jointWidth f_jointWidth] 'jointLength f_jointLength
    ['jointToEdgeEnd f_jointToEdgeEnd]
    ['jointExtension f_jointExtension]
    ['jointCornerOnly]
    [('exceptEdgeLength f_edgeLength ['prl f_parallelRunLength]) ...])
    ['prl f_parallelRunLength]
    ['endToEndPRL f_endToEndPRL]
    ['endToJointPRL f_endToJointPRL]
    ['jointToEndPRL f_jointToEndPRL]
    ['jointToJointPRL f_jointToJointPRL]
    ['sideToEndPRL f_sideToEndPRL]
    ['sideToJointPRL f_sideToJointPRL]
    ['jointToSidePRL f_jointToSidePRL]
    ['sideToSidePRL f_sideToSidePRL]
    ['sideLength f_sideLength]
    ['sideExtension f_sideExtension]
    ['sideToSideLength f_sideToSideLength]
    ['sideToSideExtension f_sideToSideExtension]
    ['sideEdgeLength f_sideEdgeLength]]
    (
      (endToEnd f_endSpacing1 f_endSpacing2)
      (jointToEnd f_jointSpacing f_endSpacing)
      (endToJoint f_endSpacing f_jointSpacing)
      (jointToJoint f_jointSpacing1 f_jointSpacing2)
      [(sideToEnd f_sideSpacing f_endSpacing)
       (sideToJoint f_sideSpacing f_jointSpacing)
        [(jointToSide f_jointSpacing f_sideSpacing)
         (sideToSide f_sideSpacing1 f_sideSpacing2)
        ]
      ]
    )
  )
) ;spacings
```

Defines the minimum spacing requirement for a wire with two neighboring wires on opposite sides with projected parallel run length greater than zero. The neighboring wires are classified as EOL or as T or L joints. End spacing is the distance between an end-of-line edge (EOL) and the middle wire and joint spacing is the distance between a joint and the middle wire.

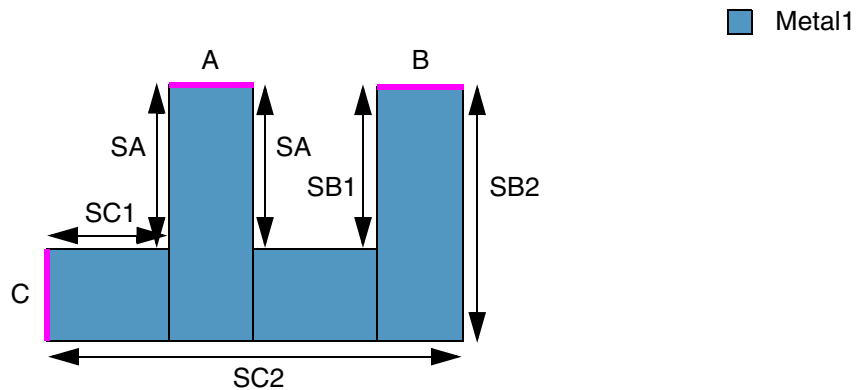
In an end-to-end or joint-to-joint scenario, either both spacing values must be greater than or equal to the smaller of the two specified spacing values, or at least one spacing must be greater than or equal to the larger of the two specified spacing values.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

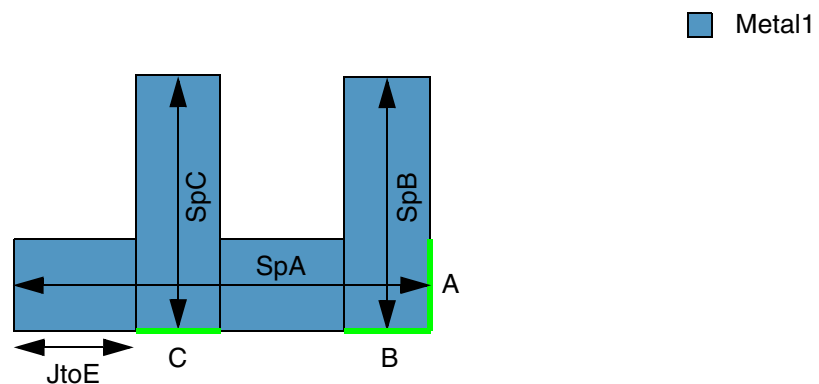
In an end-to-joint or joint-to-end scenario, both `endToJoint` and `jointToEnd` conditions must be fulfilled individually, and to fulfill a condition, either joint spacing must be greater than or equal to `jointSpacing` or end spacing must be greater than or equal to `endSpacing`.

The figure below illustrates end-of-line wires.



Edges labeled A, B, and C represent end-of-line edges and SA , SB_n , and SC_n represent the lengths of the sides that contain the respective end-of-line edge. The constraint applies if an end-of-line edge is less than `eolWidth` wide and the length of each of the two sides containing the end-of-line edge is greater than or equal to `length`.

The figure below illustrates joints.



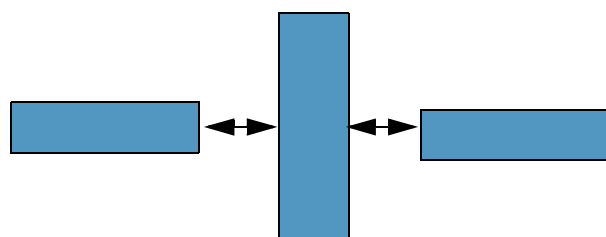
Edges labeled A, B, and C are joints if their width is less than `eolWidth` or `jointWidth` and the span lengths SpA , SpB , and SpC , respectively, are greater than `jointLength`. The constraint applies only if the distance `JtoE`, measured from the joint to the end of the edge that contains the joint, is less than or equal to `jointToEndEdge`. A joint need not necessarily be a T or L joint; a joint can be any edge that fulfills the conditions listed above.

Virtuoso Technology Data Constraints Reference

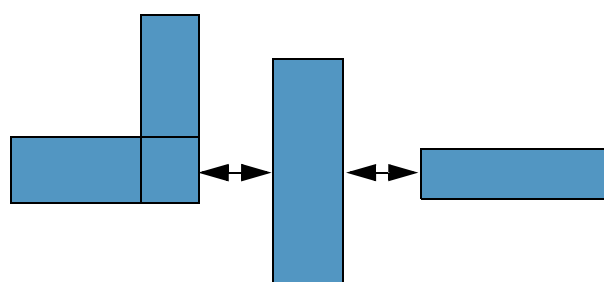
Spacing Constraints (One Layer)

The figures below illustrate the four different types of spacing. The width of the middle wire must be less than *width*.

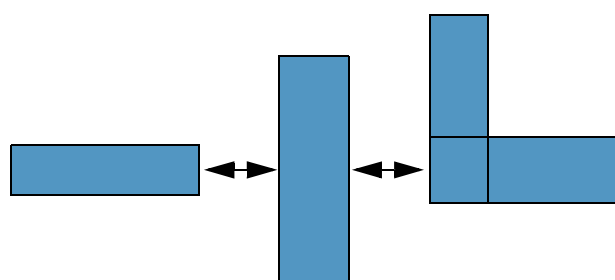
■ Metal1



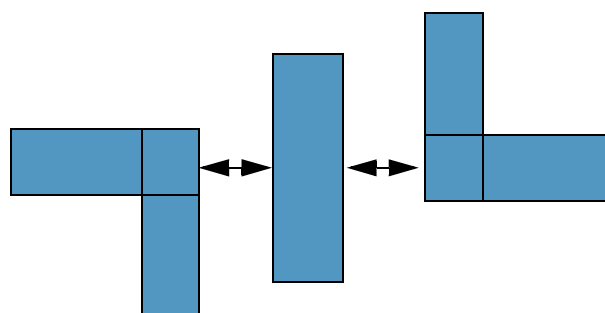
End-to-end spacing



Joint-to-end spacing



End-to-joint spacing

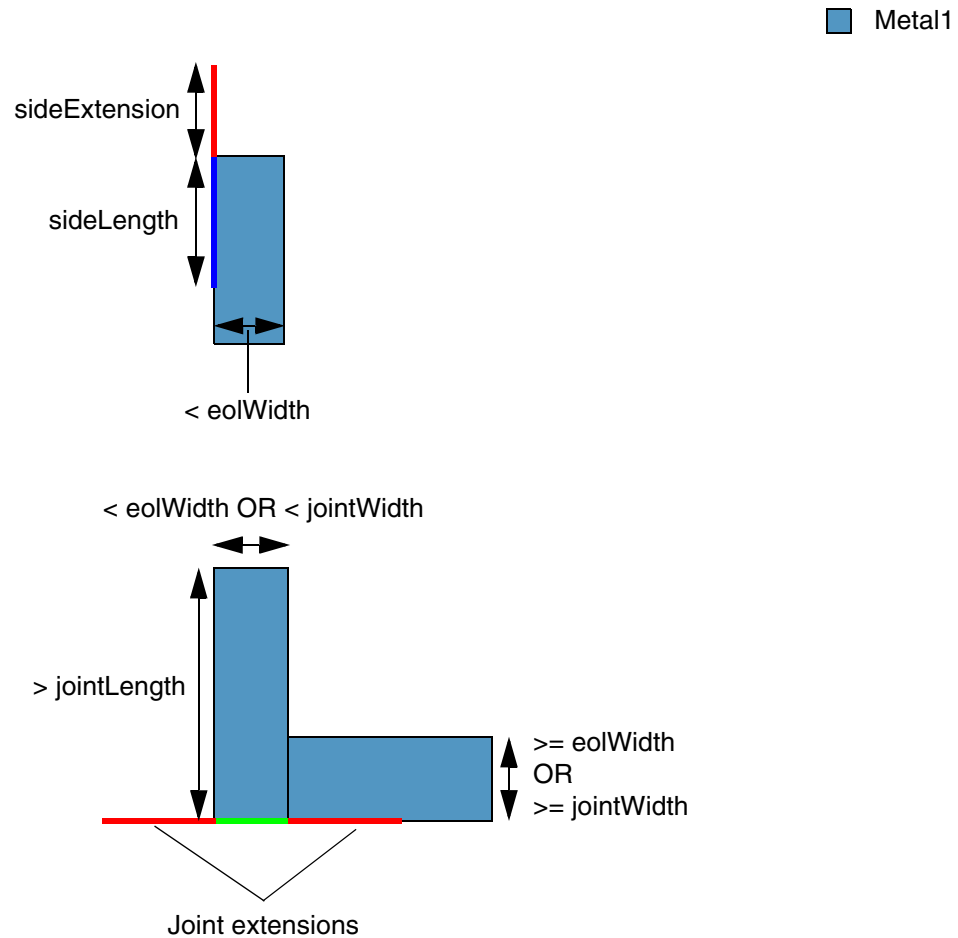


Joint-to-joint spacing

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

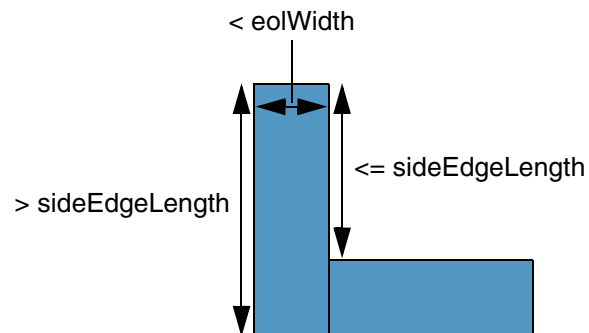
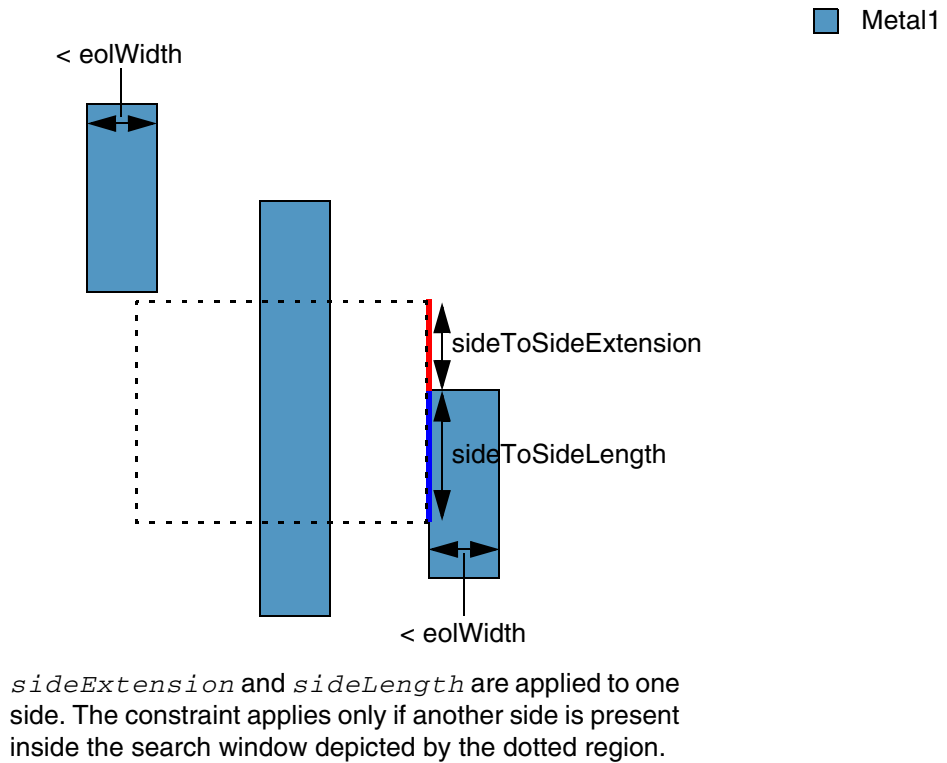
The following figures illustrate side and joint extensions. The green edge depicts a joint.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The first figure below illustrates a side-to-side spacing scenario and the second figure illustrates the criterion for determining a side edge when 'sideEdgeLength' is specified.



Only the left edge is greater than *sideEdgeLength* and qualifies as a side.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>endToEnd f_endSpacing1 f_endSpacing2</code>	Either both spacing values must be greater than or equal to the smaller of the two specified spacing values, or at least one spacing must be greater than or equal to the larger of the two specified spacing values.
<code>jointToEnd f_jointSpacing f_endSpacing</code>	The joint spacing must be at least <i>jointSpacing</i> and the end spacing must be at least <i>endSpacing</i>. Both <code>endToJoint</code> and <code>jointToEnd</code> spacing must be fulfilled individually.
<code>endToJoint f_endSpacing f_jointSpacing</code>	The end spacing must be at least <i>endSpacing</i> and the joint spacing must be at least <i>jointSpacing</i>. Both <code>endToJoint</code> and <code>jointToEnd</code> spacing must be fulfilled individually.
<code>jointToJoint f_jointSpacing1 f_jointSpacing2</code>	Either both spacing values must be greater than or equal to the smaller of the two specified spacing values, or at least one spacing must be greater than or equal to the larger of the two specified spacing values.
<code>sideToEnd f_sideSpacing f_endSpacing</code>	The spacing between a side that satisfies '<i>sideLength</i>' and the middle wire must be at least <i>sideSpacing</i> and between an end-of-line edge and the middle wire must be at least <i>endSpacing</i>.
<code>sideToJoint f_sideSpacing f_jointSpacing</code>	The spacing between a side that satisfies '<i>sideLength</i>' and the middle wire must be at least <i>sideSpacing</i> and between a joint and the middle wire must be at least <i>jointSpacing</i>.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`jointToSide f_jointSpacing f_sideSpacing`

The spacing between a joint and the middle wire must be at least *jointSpacing* and between a side that satisfies *'sideLength* and the middle wire must be at least *sideSpacing*.

`sideToSide f_sideSpacing1 f_sideSpacing2`

The spacing of the sides of neighboring wires from the middle wire must be at least *sideSpacing1* and *sideSpacing2*. Any defined global parallel run length is ignored when checking side-to-side spacing.

Parameters

`'width f_width` The constraint applies only if the middle wire has width less than this value.

`'eolWidth f_eolWidth`

The constraint applies only if the end-of-line width of the neighboring wires is less than this value.

`'length f_length` The constraint applies only if the length of both sides of a neighboring wire is greater than or equal to this value.

The neighboring wire may be a joint if its length is less than this value along any one side.

`'sameMask` (Virtuoso Advanced Node for Layout Only) The constraint applies only if all three wires are on the same mask.

`'jointWidth f_jointWidth`

If the width of an end-of-line edge of a neighboring wire is less than this value, the end-of-line edge is a joint.

`'jointLength f_jointLength`

If the width of an end-of-line edge of a neighboring wire is less than *jointWidth* or *eolWidth* and its span length is greater than this value, it is a joint.

`'jointToEdgeEnd f_jointToEdgeEnd`

The constraint applies only if the distance measured from the joint to the end of the edge that contains the joint is less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'jointExtension f_jointExtension`

The joint is extended by this value on both sides.

`'jointCornerOnly` The joint must form a joint corner, which is a convex corner consisting of two consecutive joints.

`'exceptEdgeLength f_exceptEdgeLength ['prl f_parallelRunLength]`

The constraint does not apply if *length* or *jointLength* is greater than or equal to *exceptEdgeLength* and the projected parallel run length is less than or equal to *parallelRunLength*.

`'prl f_parallelRunLength`

The constraint applies only if the projected parallel run length between the wires on opposite sides is greater than this value.

Note: This global parallel run length value applies if parallel run length is not specified for a given end, joint, and side combination.

`'endToEndPRL f_endToEndPRL`

The constraint applies only if the parallel run length for end-to-end scenario is greater than this value.

`'endToJointPRL f_endToJointPRL`

The constraint applies only if the parallel run length for end-to-joint scenario is greater than this value.

`'jointToEndPRL f_jointToEndPRL`

The constraint applies only if the parallel run length for joint-to-end scenario is greater than this value.

`'jointToJointPRL f_jointToJointPRL`

The constraint applies only if the parallel run length for joint-to-joint scenario is greater than this value.

`'sideToEndPRL f_sideToEndPRL`

The constraint applies only if the parallel run length for side-to-end scenario is greater than this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'sideToJointPRL f_sideToJointPRL`

The constraint applies only if the parallel run length for side-to-joint scenario is greater than this value.

`'jointToSidePRL f_jointToSidePRL`

The constraint applies only if the parallel run length for joint-to-side scenario is greater than this value.

`'sideToSidePRL f_sideToSidePRL`

The constraint applies only if the parallel run length for side-to-side scenario is greater than this value.

`'sideLength f_sideLength`

The length of the edges of a neighboring wire containing the end-of-line edge must be greater than or equal to this value in a side-to-end and side-to-joint scenario.

If `'sideLength` is defined, then `sideToEnd` and `sideToJoint` must also be defined.

`'sideExtension f_sideExtension`

One side of a neighboring wire that satisfies `'sideLength` is extended by this value in the direction beyond the end-of-line edge in a side-to-end and side-to-joint scenario. The other side has zero extension.

The `'sideLength` and `'sideExtension` parameters together define a search window.

`'sideToSideLength f_sideToSideLength`

The length of the edges of the wires containing the end-of-line edge must be greater than or equal to this value in a side-to-side scenario.

`'sideToSideExtension f_sideToSideExtension`

One side of a neighboring wire that satisfies `'sideToSideLength` is extended by this value in the direction beyond the end-of-line edge in a side-to-side scenario. The other side has zero extension.

The `'sideToSideLength` and `'sideToSideExtension` parameters together define a search window. The constraint applies only if a side is present inside the search window.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'sideEdgeLength f_sideEdgeLength`

The length of an edge containing a joint must be greater than this value for the edge to be considered a side.

Examples

- Example 1: minOppSpanSpacing with width, eolWidth, jointLength, and exceptEdgeLength
- Example 2: minOppSpanSpacing with width, eolWidth, jointLength, sideLength, and prl

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 1: minOppSpanSpacing with width, eolWidth, jointLength, and exceptEdgeLength

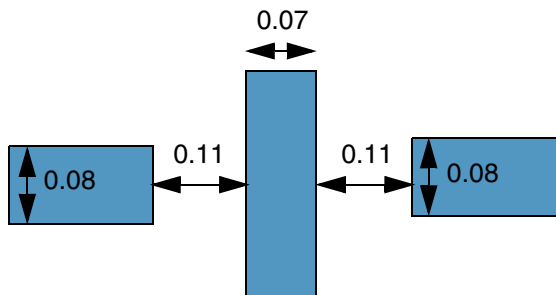
The constraint applies if the following conditions are satisfied:

- Width of the middle wire is less than 0.08
- End-of-line width of an end wire is less than 0.1
- Span length of the joint is greater than 0.15

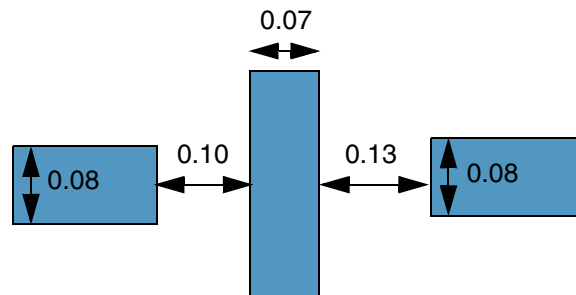
The constraint does not apply if the span length of the joint is greater than 0.08 and the parallel run length between two neighboring wires is less than 0.03.

```
spacings(  
  ( minOppSpanSpacing "Metal1"  
    'width 0.08 'eolWidth 0.1  
    'jointLength 0.15  
    'exceptEdgeLength 0.08  
    'prl 0.03  
    (  
      (endToEnd 0.11 0.13)  
      (jointToEnd 0.12 0.10)  
      (endToJoint 0.12 0.10)  
      (jointToJoint 0.12 0.16)  
    )  
  )  
) ;spacings
```

■ Metal1



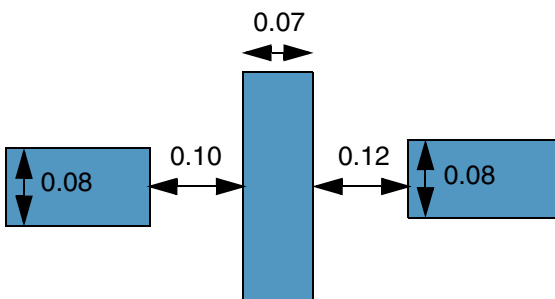
a) PASS. The width of the middle wire is 0.07 (<0.08) and the end-of-line width of the two neighboring wires is 0.08 (<0.1). Therefore, the constraint applies. The spacing on both sides is 0.11, which is equal to the smaller of the two specified spacing values.



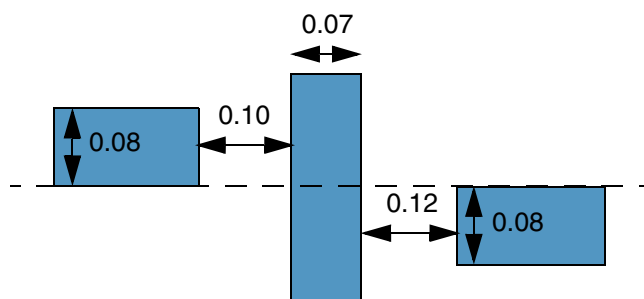
b) PASS. The spacing on the right is 0.13, which is equal to the larger of the two specified spacing values.

Virtuoso Technology Data Constraints Reference

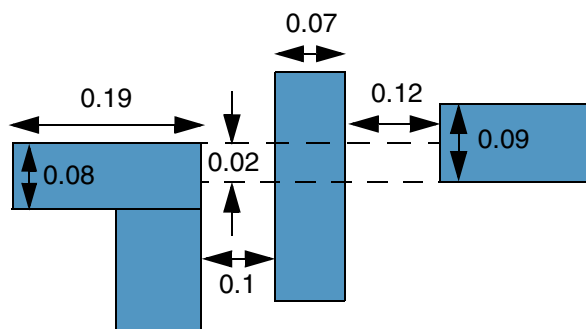
Spacing Constraints (One Layer)



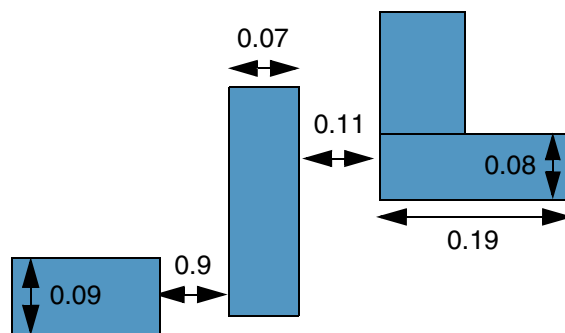
c) FAIL. The spacing on the left is 0.10, which is less than the smaller of the two specified spacing values.



d) The constraint does not apply because the projected parallel run length between the wires on the opposite sides is zero.



e) The constraint does not apply because the length of the joint is 0.19 (>0.08) and the parallel run length is 0.02 (<0.03).



f) FAIL. End-to-joint spacing requires at least 0.12 end spacing and 0.10 joint spacing, of which joint spacing is satisfied. Joint-to-end spacing requires at least 0.10 end spacing and 0.12 joint spacing; of these, none is satisfied.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

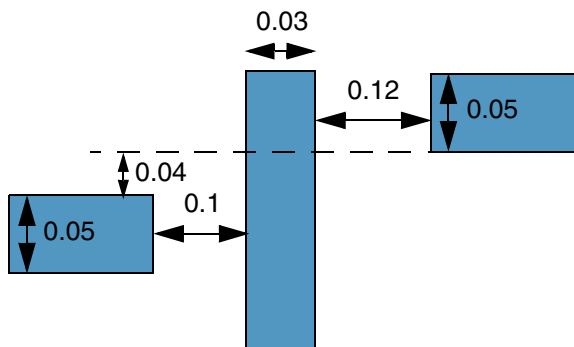
Example 2: minOppSpanSpacing with width, eolWidth, jointLength, sideLength, and prl

The constraint applies if the following conditions are satisfied:

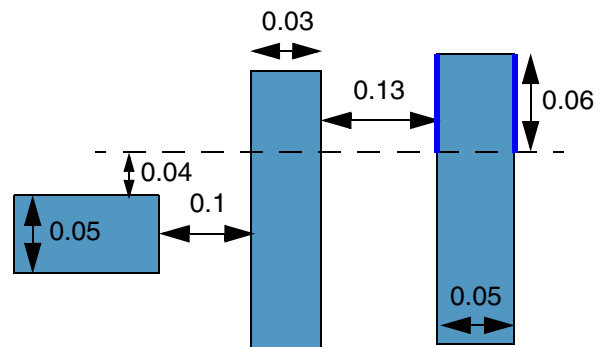
- Width of the middle wire is less than 0.08
- End-of-line width of an end wire is less than 0.1
- Span length of the joint is greater than 0.15
- Length of sides containing an end-of-line edge is at least 0.06
- Parallel run length between two neighboring wires is greater than -0.05

```
spacings (
  ( minOppSpanSpacing "Metal1"
    'width 0.08 'eolWidth 0.1
    'jointLength 0.15
    'sideLength 0.06
    'prl -0.05
    (
      (endToEnd 0.11 0.13)
      (jointToEnd 0.12 0.10)
      (endToJoint 0.12 0.10)
      (jointToJoint 0.12 0.16)
      (sideToEnd 0.14 0.15)
      (sideToJoint 0.14 0.15)
    )
  )
) ;spacings
```

■ Metal1



a) FAIL. The constraint applies because the parallel run length between the two wires is -0.04 (> -0.05). However, the end spacing on the left is 0.1 (< 0.11) and on the right is 0.12 (< 0.13).



b) FAIL. The thick blue edges are sides of an end-of-line edge, but the distance of the side from the middle wire is 0.13 (< 0.14). Additionally, the distance of the end-of-line edge from the middle wire is 0.1 (< 0.15).

Virtuoso Technology Data Constraints Reference

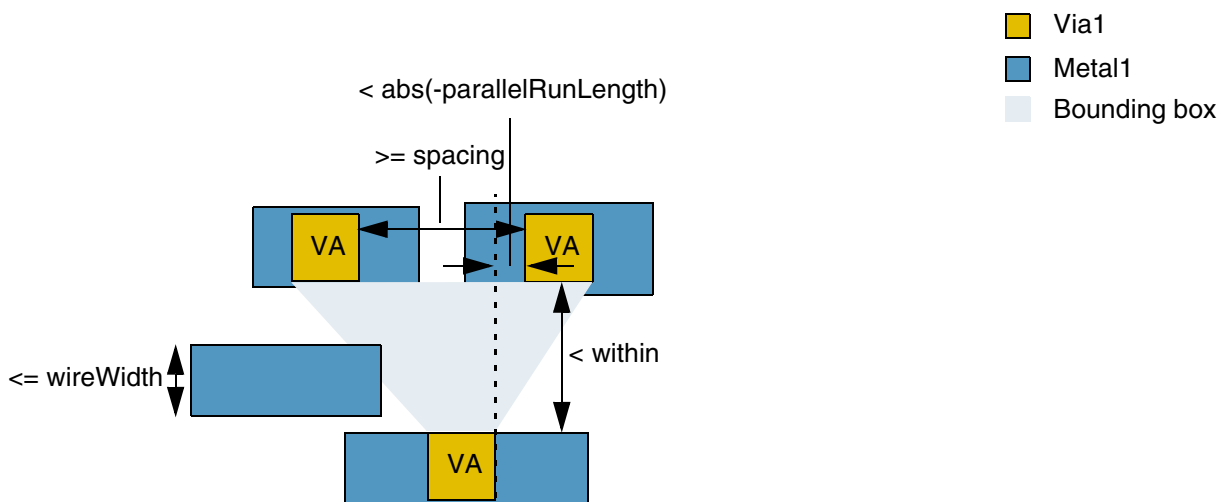
Spacing Constraints (One Layer)

minOrthogonalSpacing

```
orderedSpacings (
  ( minOrthogonalSpacing tx_cutLayer tx_metalLayer
    ['cutClass {f_width | (f_width f_length) | t_name}
    ['toAll]
    | 'cutClassList l_cutClassNames]
    'prl f_parallelRunLength 'within f_within
    'width f_wireWidth
    ['metalWithin (g_metalPr1 g_metalWithin)]
    ['horizontal | 'vertical]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ]
  f_spacing
)
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Only) Specifies that the spacing between two exactly aligned via cuts must be greater than or equal to *spacing* if the following conditions are met:

- A neighboring via cut is present in the orthogonal direction at a distance less than *within* from the via cuts and has parallel run length greater than *parallelRunLength* with the via cuts.
- A wire with width less than or equal to *wireWidth* on *metalLayer* overlaps the bounding box formed by the via cut edges that face each other, as shown below.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the exactly aligned via cuts must be greater than or equal to this value.

Parameters

<code>'cutClass {f_width (f_width f_length) t_name}</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class Otherwise, the constraint applies to via cuts of any cut class.
<code>'toAll</code>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies when at least one cut is of the specified cut class even if the other cuts are of any cut classes.
<code>'cutClassList l_cutClassNames</code>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if cuts belong to the specified cut classes. Each list entry is a doublet with values indicating the dimensions of a cut.
<code>'prl f_parallelRunLength</code>	The constraint applies only if the parallel run length between the exactly aligned via cuts and the neighboring via cut, which is at a distance less than <i>within</i> from them, is greater than this value.
<code>'within f_within</code>	

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The constraint applies only if the distance between the exactly aligned via cuts and the neighboring via cut is less than this value.

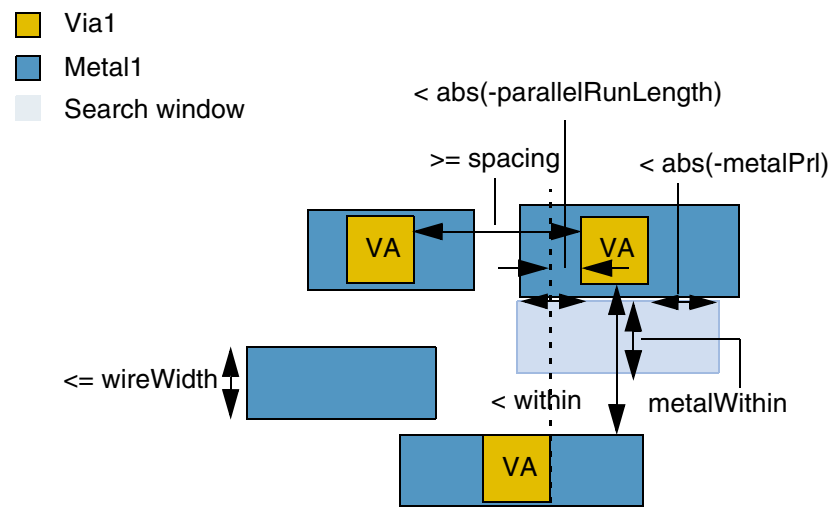
`'width f_wireWidth`

The constraint applies only if the width of the *metalLayer* wire is less than or equal to this value.

`'metalWithin (g_metalPr1 g_metalWithin)`

Specifies that the search window of a metal neighbor wire is formed by *metalPr1* (first int) and is extended from the cut with a cut neighbor by *metalWithin* (second int).

The following figure illustrates *metalWithin* and *metalPr1*.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'horizontal | 'vertical

The spacing applies only in the specified direction.

'insideLayers (*tx_layer1 tx_layer2 ... tx_layerN*)
| 'outsideLayers (*tx_layer1 tx_layer2 ... tx_layerN*)

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

'insidePurposes (*t_purpose1 t_purpose2 ... t_purposeN*)

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

'outsidePurposes (*t_purpose1 t_purpose2 ... t_purposeN*)

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Example

Specifies that the spacing between Via2 and Metal3 must be greater than or equal to 0.07 for the VA cut class when the parallel run length is 0.05, the width of the metal layer wire is less than or equal to 0.08, the search window is between 0.03 to 0.04, and the distance between the aligned via cuts and the neighboring via cut is less than 0.06.

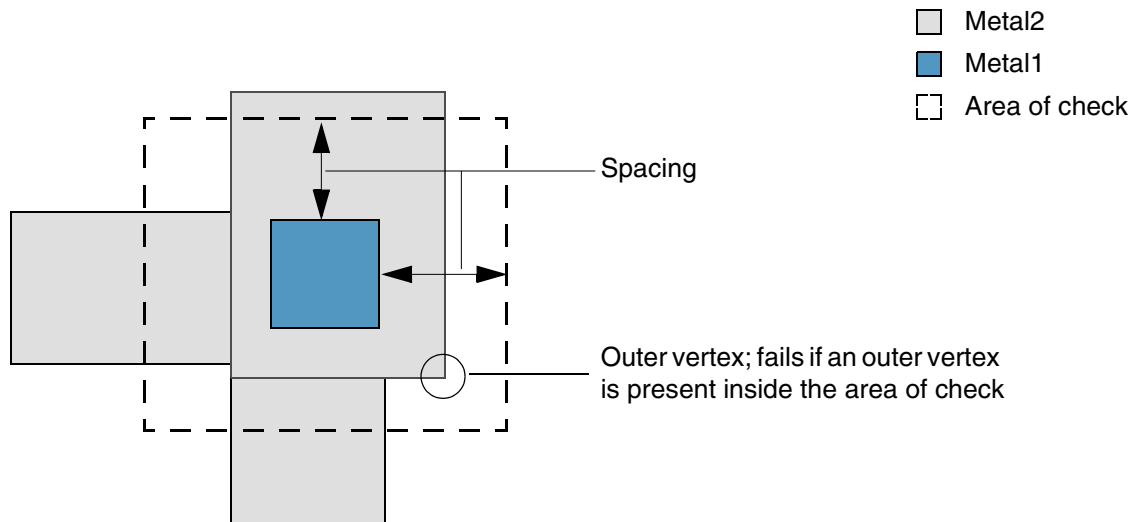
```
orderedSpacings (  
  (minOrthogonalSpacing "Via2" "Metal3"  
    'cutClass "VA" 'prl 0.05 'width 0.08 'metalWithin (0.03 0.04)  
    'within 0.06 0.07)  
) ;orderedSpacings
```

minOuterVertexSpacing

```
orderedSpacings (
    ( minOuterVertexSpacing tx_layer1 tx_layer2
      f_spacing
    )
) ;orderedSpacings
```

Specifies the minimum spacing between a shape on *layer1* and an outer vertex of the enclosing shape on *layer2*. An outer vertex is defined as a convex corner.

The spacing is checked using Manhattan measure type. A violation occurs if a convex corner of a *layer2* shape is found inside the area of check defined by *spacing*, as shown below.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum required spacing.

Parameters

None

Example

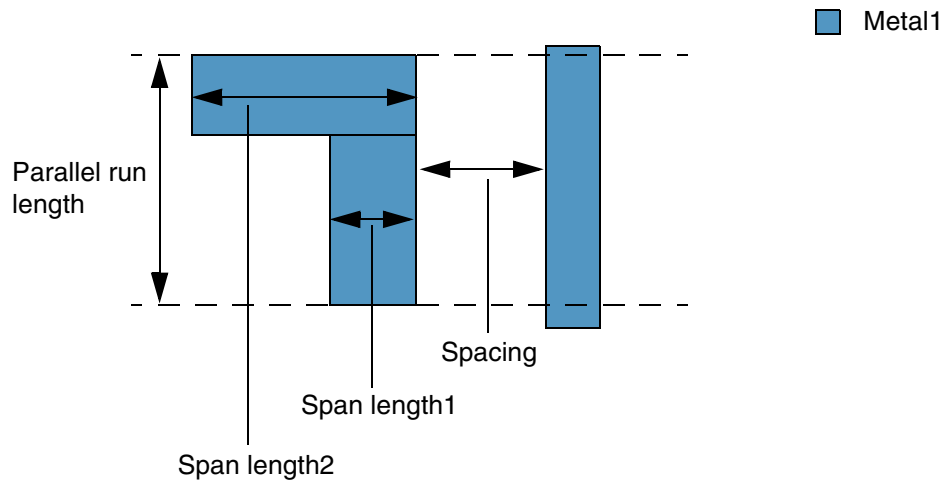
The spacing of a Metal2 shape from an outer vertex (convex corner) of a Metal1 shape must be at least 0.32.

```
orderedSpacings (
  ( minOuterVertexSpacing "Metal2" "Metal1"
    0.32
  )
) ;orderedSpacings
```

minParallelSpanSpacing

```
spacingTables (
  ( minParallelSpanSpacing tx_layer
    (( "span" nil nil "span" nil nil )
      'length f_length
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum spacing between two shapes on the same layer based on their span lengths. The span length, which is defined as the length of the shape perpendicular to the edge corresponding to the parallel run length, can be different for different subsections of the shape. The parallel run length is measured as the continuous summed parallel run length between shapes. The constraint applies only if the parallel run length between the two shapes is greater than the specified value.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"span" nil nil "span" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of a <i>table</i> row is as follows: (<i>f_index1</i> <i>f_index2</i>) <i>f_value</i> where, ■ <i>f_index1</i> is the span length of the first shape. ■ <i>f_index2</i> is the span length of the second shape. ■ <i>f_value</i> is the minimum required spacing between shapes. To determine the required minimum spacing, you need to locate the last row for which all span lengths of the first shape are greater than the index value (<i>index1</i>). Similarly, you need to locate the last column for which all span lengths of the second shape are greater than the index value (<i>index2</i>). The required spacing lies at the intersection of the row and column. Type: A symmetric 2-D table specifying floating-point spacing values. The number of spacing values in each row must be equal to the number of rows. The span length and spacing values must increase from top to bottom and from left to right.

Parameters

<i>f_length</i>	The constraint applies only if the parallel run length between the two shapes is greater than this value. Turning corners can break up the parallel run length, resulting in inaccurate calculations.
<i>f_default</i>	The spacing value to be used when no table entry applies.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The spacing requirement between two shapes with span lengths represented by `index1` and `index2` is as given in the table below:

Spacing	<code>index2 ></code>	0.00	0.10	0.20
<code>index1 ></code>				
0.00		0.10	0.15	0.20
0.10		0.15	0.17	0.23
0.20		0.20	0.23	0.25

Virtuoso Technology Data Constraints Reference

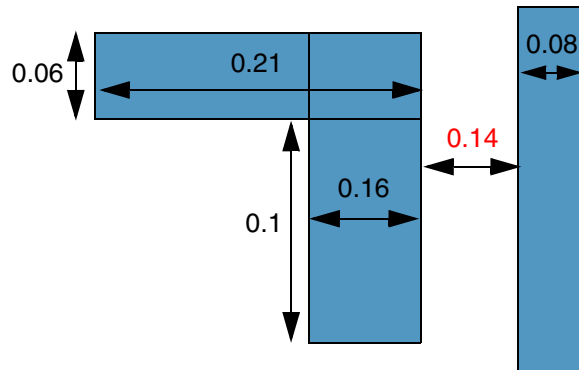
Spacing Constraints (One Layer)

```
spacingTables(
  ( minParallelSpanSpacing "Metal1"
    (( "span" nil nil "span" nil nil )
      'length 0.15
    )
  )
  (
    (0.00 0.00) 0.10
    (0.00 0.10) 0.15
    (0.00 0.20) 0.20

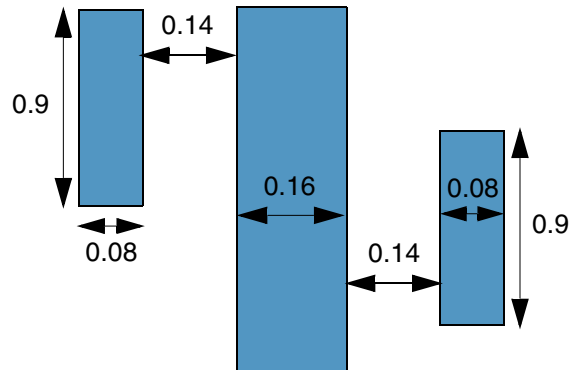
    (0.10 0.00) 0.15
    (0.10 0.10) 0.17
    (0.10 0.20) 0.23

    (0.20 0.00) 0.20
    (0.20 0.10) 0.23
    (0.20 0.20) 0.25
  )
) ;spacingTables
```

■ Metal1



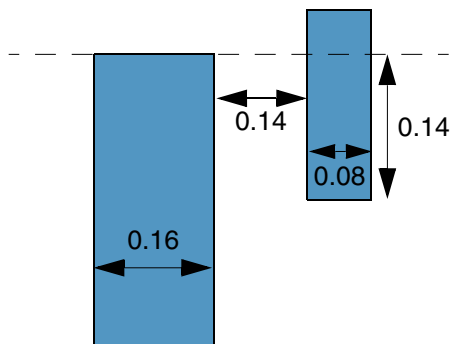
a) FAIL. Both span lengths of the shape on the left are greater than 0.1, and the span length of the shape on the right is greater than 0. Additionally, the parallel run length between the shapes is 0.16 (> 0.15). This requires a spacing of 0.15, but the spacing is only 0.14.



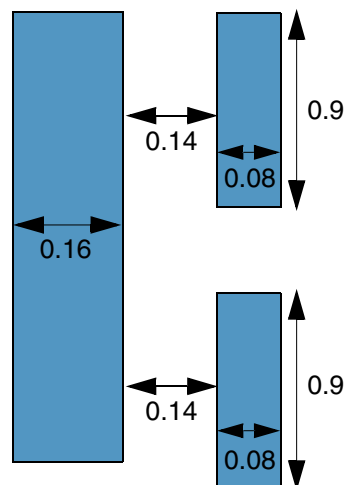
b) The constraint does not apply because the parallel run length with the shapes to the left and right is only 0.9 (< 0.15). Parallel run lengths of shapes on opposite sides do not add up.

Virtuoso Technology Data Constraints Reference

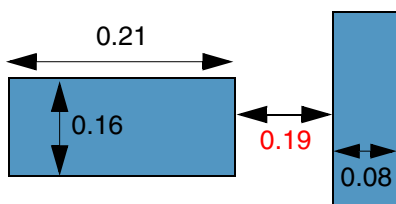
Spacing Constraints (One Layer)



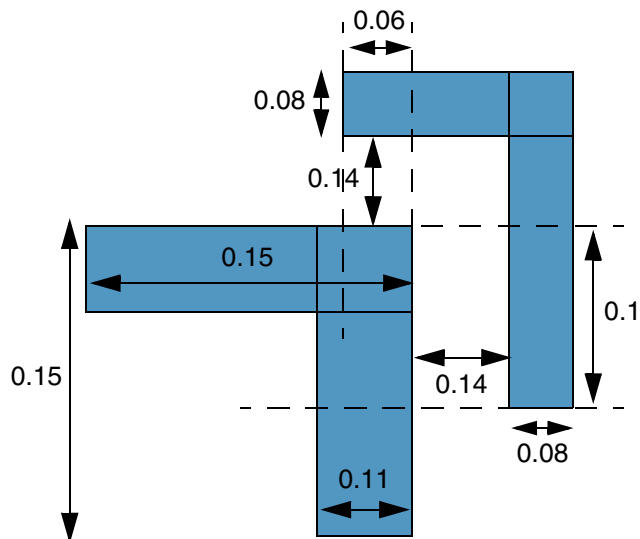
c) The constraint does not apply because the parallel run length between the two shapes is only 0.14 (<0.15).



d) The constraint does not apply because the parallel run length is not continuous and individual parallel run lengths are both less than 0.15.



e) FAIL. The parallel run length between the two shapes is 0.16 (>0.15) and the span lengths of the two shapes are 0.21 and 0.08. This requires a spacing of 0.20. However, spacing is only 0.19.



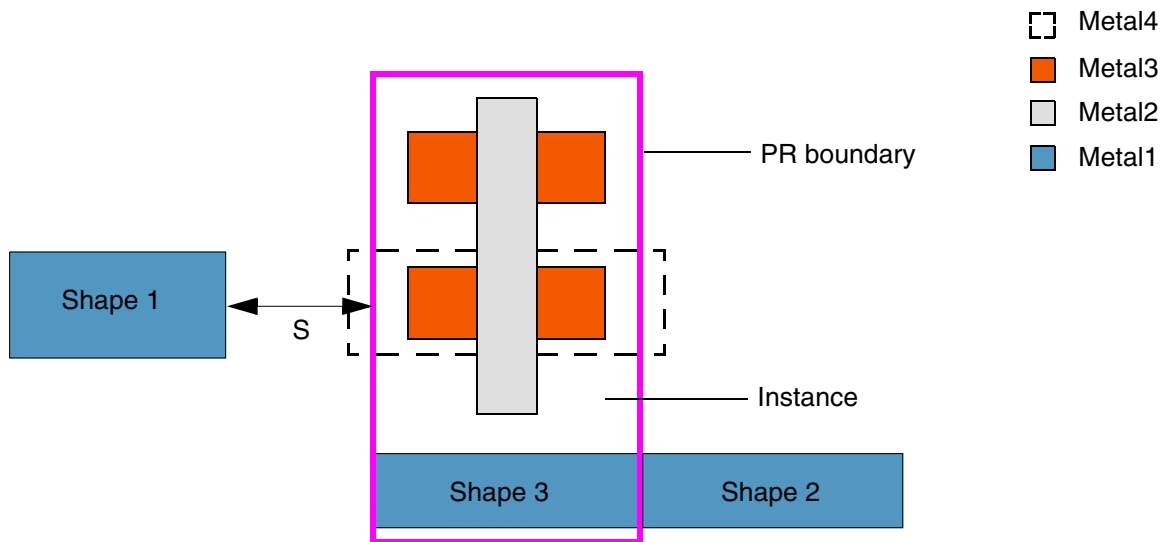
f) The constraint does not apply because the top and side parallel run lengths are both less than 0.15.

minPRBoundaryExteriorHalo

```
spacings (  
    ( minPRBoundaryExteriorHalo tx_layer  
      f_spacing ['coincidentAllowed']  
    )  
); spacings
```

Specifies the minimum spacing on a layer between a shape and the PR boundary of an instance in the current level of the hierarchy.

In the figure below, the minimum spacing between Shape 1 and the PR boundary of the instance is S . Shape 2 coincides with the PR boundary and would not cause a violation if 'coincidentAllowed' is specified. The PR boundary need not be the same as the bounding box of the instance, as is the case below, where Metal4 in the master of the instance projects beyond the PR boundary.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shape and the PR boundary must be greater than or equal to this value.

Parameters

'coincidentAllowed	The shape can coincide with the PR boundary. Otherwise, the spacing requirement must be met. Type: Boolean

Example

The spacing between a Metal1 shape and the PR boundary of an instance in the current level of the hierarchy must be at least 1.0.

```
spacings(  
  ( minPRBoundaryExteriorHalo "Metal1"  
    1.0  
  )  
) ;spacings
```


minPRBoundaryInteriorHalo

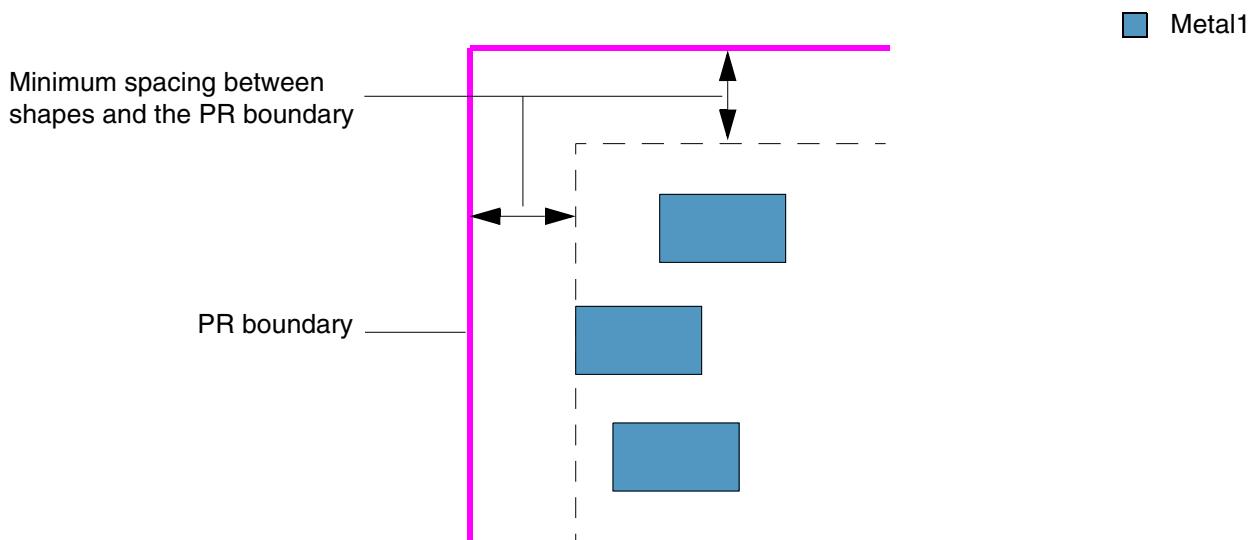
```

spacings (
    ( minPRBoundaryInteriorHalo tx_layer
      ['horizontal | 'vertical]
      ['stepSize f_stepSize]
      ['exceptLayer tx_exceptLayer 'exceptWidth f_exceptWidth]
      f_spacing ['coincidentAllowed]
    )
) ;spacings

```

Specifies the minimum spacing between a shape and the enclosing PR boundary on a layer.

The constraint can optionally restrict the allowed spacing to integer multiples of a step value and can also be optionally applied only in the specified direction.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shape and the PR boundary must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'horizontal | 'vertical`

(Virtuoso Advanced Node for Layout Standard) The direction in which spacing is measured. If direction is not specified, spacing is measured in any direction.

Type: Boolean

`'stepSize f_stepSize`

(Virtuoso Advanced Node for Layout Standard) The allowed spacing (S_{legal}) is restricted to the following values:

$$S_{\text{legal}} = S_{\text{min}} + n * \text{stepSize}$$

where, S_{min} is the minimum spacing requirement.

`'exceptLayer tx_exceptLayer 'exceptWidth f_exceptWidth`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to shapes that overlap (fully or partially) a shape on the `tx_exceptLayer` layer whose width is less than or equal to `f_exceptWidth`.

`'coincidentAllowed` The shape can coincide with the PR boundary. Otherwise, the spacing requirement must be met.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Examples

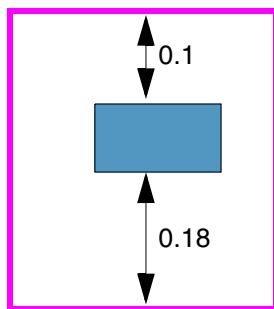
- Example 1: minPRBoundaryInteriorHalo with stepSize
- Example 2: minPRBoundaryInteriorHalo with exceptLayer and exceptWidth

Example 1: minPRBoundaryInteriorHalo with stepSize

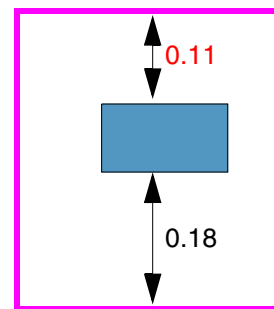
The vertical spacing between a Metal1 shape and the enclosing PR boundary must be at least 0.1 or equal to the sum of the constraint value (0.1) and an integer multiple of 0.04.

```
spacings(  
  ( minPRBoundaryInteriorHalo "Metal1"  
    'vertical  
    'stepSize 0.04  
    0.1  
  )  
); spacings
```

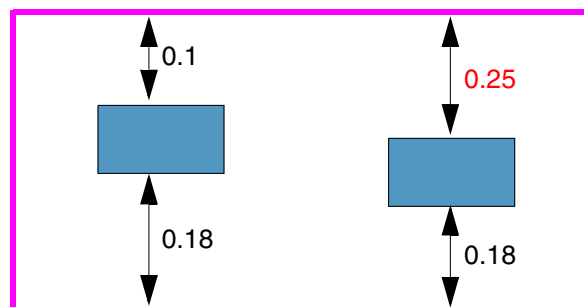
■ Metal1
■ PR boundary



a) PASS. The upper vertical spacing is equal to 0.1 and the bottom vertical spacing of 0.18 is equal to 0.1 plus an integer multiple of 0.04 ($0.1 + 2 \times 0.04 = 0.18$).



b) FAIL. The upper vertical spacing of 0.11 is not equal to the 0.1 plus an integer multiple of 0.04.



c) FAIL. The vertical spacing of 0.25 is not equal to the 0.1 plus an integer multiple of 0.04.

Virtuoso Technology Data Constraints Reference

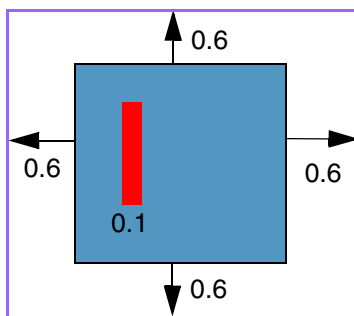
Spacing Constraints (One Layer)

Example 2: *minPRBoundaryInteriorHalo* with *exceptLayer* and *exceptWidth*

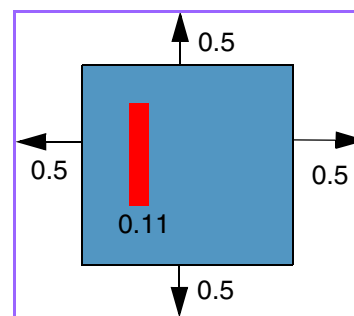
The spacing between a Metal1 shape and the PR boundary of an instance in the current level of the hierarchy must be at least 1.0 except when it overlaps a Poly shape whose width is less than or equal to 0.1.

```
spacings(  
  ( minPRBoundaryInteriorHalo "Metal1"  
    'exceptLayer "Poly"  
    'exceptWidth 0.1  
    1.0  
  )  
) ;spacings
```

■ Poly
■ Metal1
□ PR Boundary



FAIL. The constraint applies because the Metal1 shape overlaps a Poly shape with width less than or equal to 0.1. It fails because the spacing is less than 1.



PASS. The constraint does not apply because Metal1 overlaps only those Poly shapes that are wider than 0.1.

minPrlTwoSidesSpacing

```
spacings (
  ( minPrlTwoSidesSpacing tx_cutLayer
    'cutClass {f_width | (f_width f_length) | t_name}
    'otherCutClass {f_width | (f_width f_length) | t_name}
    'prlRangeOne g_prlRangeOne 'prlRangeTwo g_prlRangeTwo
    (f_prlSpacing f_nonPrlSpacing)
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies that the center-to-center spacing between two cut shapes on a layer must be greater than or equal to *prlSpacing* if the parallel run length between the cut shapes satisfies both 'prlRangeOne and 'prlRangeTwo. If the cut shapes do not satisfy 'prlRangeOne and 'prlRangeTwo, the center-to-center spacing between them must be greater than or equal to *nonPrlSpacing*. The cut shapes can belong to the same cut class or to different cut classes.

Values

<i>tx_cutLayer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_prlSpacing f_nonPrlSpacing</i>	The spacing between two cut shapes must be greater than or equal to <i>prlSpacing</i> or <i>nonPrlSpacing</i> .

Parameters

<pre>'cutClass {f_width (f_width f_length) t_name}</pre>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
'otherCutClass {f_width | (f_width f_length) | t_name}
```

The other cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

```
'prlRangeOne g_prlRangeOne 'prlRangeTwo g_prlRangeTwo
```

If the parallel run length falls in the range *prlRangeOne* in one direction and in the range *prlRangeTwo* in the perpendicular direction, the center-to-center spacing between the two cut shapes must be greater than or equal to *prlSpacing*.

Type: Floating-point values specifying a [range](#) of parallel run lengths

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

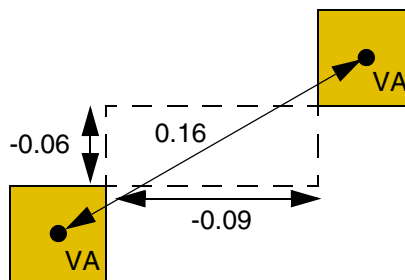
Example

The center-to-center spacing between two VA via cuts must be greater than or equal to 0.15 if the parallel run length between them falls in the range [-0.07 -0.05] in one direction and in the range [-0.1 -0.08] in the perpendicular direction.

If the parallel run length is not satisfied in one or both directions, the center-to-center spacing between the two via cuts must be greater than or equal to 0.2.

```
spacings(  
  ( minPrlTwoSidesSpacing "Via1"  
    'cutClass "VA"  
    'otherCutClass "VA"  
    'prlRangeOne "[-0.07 -0.05]" 'prlRangeTwo "[-0.1 -0.08]"  
    (0.15 0.2)  
  )  
) ;spacings
```

■ Via1



PASS. The parallel run length in one direction is -0.06, which falls in the range [-0.07 -0.05], and the parallel run length in the perpendicular direction is -0.09, which falls in the range [-0.1 -0.08]. Therefore, the center-to-center spacing between the two via cuts must be greater than or equal to 0.15, a condition that is satisfied.

minProtrusionSpacing

```
spacings (
  ( minProtrusionSpacing tx_layer
    'width f_width 'length f_length 'within f_within
    ['excludeSpacing (g_range)]
    'protrusionLength f_protrusionLength
    (
      short f_shortJogSpacing
      long f_longJogSpacing
      between f_jogToJogSpacing
    )
  )
) ;spacings
```

Specifies the minimum spacing on a layer between wires, either of which can have a protrusion, if the wide wire has width greater than *width* and shares a parallel run length greater than *length* with the neighboring wire that is at a distance less than *within* from it.

If the projected parallel run length of the protrusion with the neighboring wire is less than or equal to *protrusionLength*, the spacing between wires must be at least *shortJogSpacing*, and if the common projected parallel run length of the protrusion with the neighboring wire is greater than *protrusionLength*, the spacing between wires must be at least *longJogSpacing*.

If there are two or more protrusions, each of which has projected parallel run length less than or equal to *protrusionLength* with the neighboring wire, and spacing between the protrusion and the wire is less than *longJogSpacing*, the spacing between the protrusions in the direction of the wide wire must be greater than or equal to *jogToJogSpacing*.

Values

<i>tx_layer</i>	The layer on which the constraint is applied.
	Type: String (layer and purpose names) or Integer (layer number)

<i>short f_shortJogSpacing</i>	If the parallel run length of the protrusion in either wire with the neighboring wire is less than or equal to <i>protrusionLength</i> , spacing between wires must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`long f_longJogSpacing`

If the parallel run length of the protrusion in either wire with the neighboring wire is greater than *protrusionLength*, spacing between wires must be greater than or equal to this value.

`between f_jogToJogSpacing`

If there are two or more protrusions, each of which has projected parallel run length less than or equal to *protrusionLength* with the neighboring wire, and spacing between the protrusion and the neighboring wire is less than *longJogSpacing*, the spacing between the protrusions in the direction of the wide wire must be greater than or equal to *jogToJogSpacing*.

Parameters

`'width f_width`

The constraint applies only if the width of the wide wire is greater than this value.

`'length f_length`

The constraint applies only if the parallel run length between the two wires is greater than this value.

`'within f_within`

The constraint applies only if the distance between the wires is less than this value.

`'excludeSpacing (g_range)`

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if the distance between the two wires falls in this range.

Type: Floating-point values specifying a range of distances that are exempted

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'protrusionLength f_protrusionLength`

If the projected parallel run length of the protrusion with the neighboring wire is less than or equal to this value, the spacing between wires must be greater than or equal to *shortJogSpacing*; otherwise, spacing must be greater than or equal to *longJogSpacing*.

Example

The spacing between two wires, either of which can have a protrusion, must be as follows if the wide wire has width greater than 0.4 and shares a parallel run length greater than 0.2 with a neighboring wire that is at a distance less than 0.5 from it:

- At least 0.08 if the protrusion width is less than or equal to 0.2
- At least 0.13 if the protrusion width is greater than 0.2

The spacing between two protrusions must be at least 0.7 if each of them has projected parallel run length less than or equal to 0.2 with the neighboring wire when the spacing between a protrusion and the neighboring wire is less than 0.13.

If the wide wire has width greater than 0.8 and shares a parallel run length greater than 0.3 with a neighboring wire that is at a distance less than 0.5 from it, then the spacing between the two wires, either of which can have a protrusion, must be as follows:

- At least 0.08 if the protrusion width is less than or equal to 0.2
- At least 0.15 if the protrusion width is greater than 0.2

The spacing between two protrusions must be at least 0.7 if each of them has projected parallel run length less than or equal to 0.3 with the neighboring wire when the spacing between a protrusion and the neighboring wire is less than 0.15.

Virtuoso Technology Data Constraints Reference

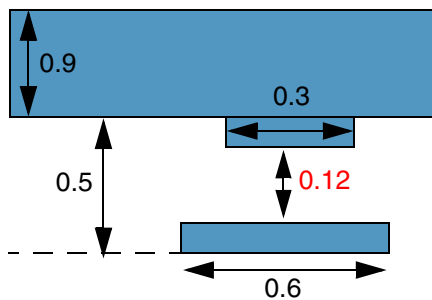
Spacing Constraints (One Layer)

```

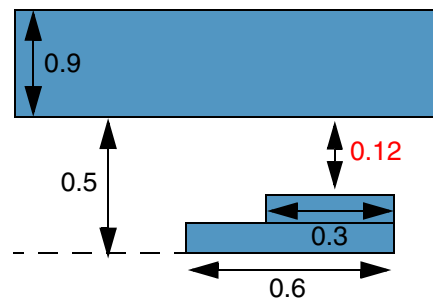
spacings (
  ( minProtrusionSpacing "Metal1"
    'width 0.4 'length 0.2 'within 0.5
    'protrusionLength 0.2
    (
      short 0.08
      long 0.13
      between 0.7
    )
  )
  ( minProtrusionSpacing "Metal1"
    'width 0.8 'length 0.3 'within 0.5
    'protrusionLength 0.2
    (
      short 0.08
      long 0.15
      between 0.7
    )
  )
) ;spacings

```

■ Metal1



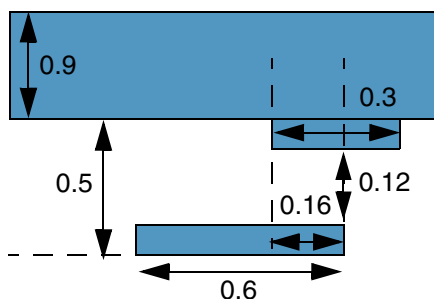
a) FAIL. All triggering conditions are met: the wide wire is 0.9 wide (>0.8); the parallel run length between the neighboring wires is 0.6 (>0.3); and the distance of the neighboring wire from the wide wire is less than 0.5. The parallel run length of the protrusion with the neighboring wire is 0.3 (>0.2); therefore, the spacing required is at least 0.15. However, the spacing is 0.12 (<0.15).



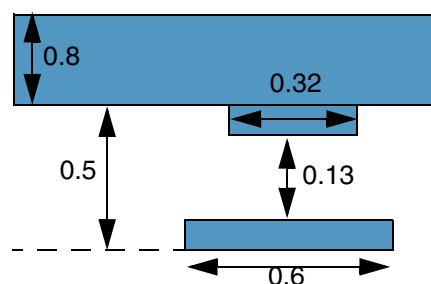
b) FAIL. All triggering conditions are met. The parallel run length of the protrusion with the neighboring wire is 0.3 (>0.2); therefore, the spacing required is at least 0.15. However, the spacing is only 0.12 (<0.15).

Virtuoso Technology Data Constraints Reference

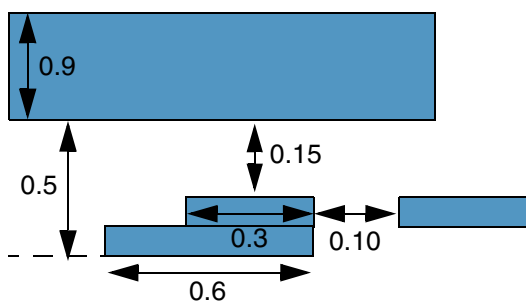
Spacing Constraints (One Layer)



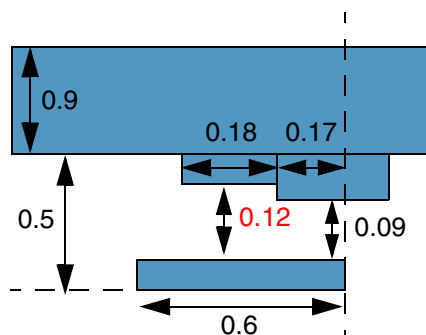
c) PASS. All triggering conditions are met. The parallel run length of the protrusion with the neighboring wire is 0.16 (<0.2); therefore, the spacing required is at least 0.08, which is met.



d) PASS. The first `minProtrusionSpacing` constraint applies because the width of the wide wire is 0.8 (which is greater than 0.4, but not greater than 0.8). The parallel run length of the protrusion with the neighboring wire is 0.32 (<0.2); therefore, the spacing required is at least 0.13, which is met.



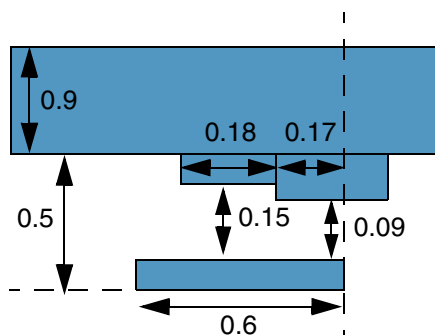
e) PASS. All triggering conditions are met. The parallel run length of the protrusion with the neighboring wire is 0.3 (>0.2); therefore, the spacing required is at least 0.15, which is met.



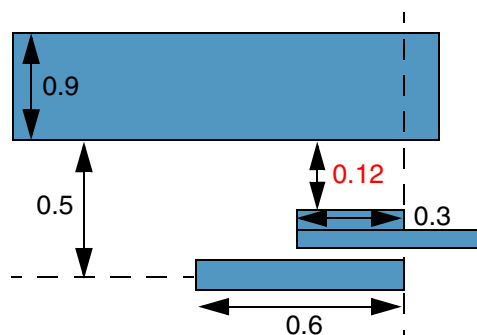
f) FAIL. All triggering conditions are met. The combined parallel run length of the protrusion with the neighboring wire is 0.35 ($=0.18+0.17$); therefore, the spacing required is at least 0.15, which is not met.

Virtuoso Technology Data Constraints Reference

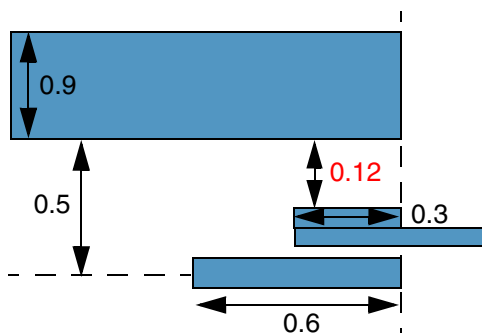
Spacing Constraints (One Layer)



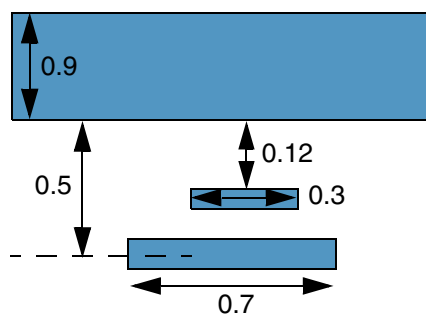
g) PASS. All triggering conditions are met. The combined parallel run length of the protrusion with the neighboring wire is 0.35 ($=0.18+0.17$); therefore, the spacing required is at least 0.15, which is met. The two 0.18 and 0.17 wide segments are both less than 0.2. This requires a spacing of at least 0.08, which is also met.



h) FAIL. The protrusion is 0.3 wide and requires a spacing of at least 0.15, a condition that is not met.



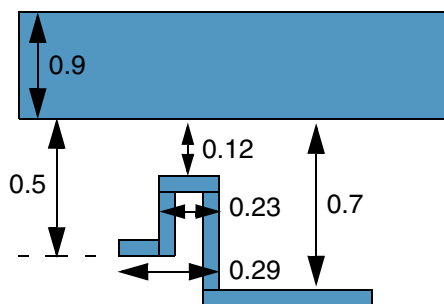
i) FAIL. The protrusion is 0.3 wide and requires spacing of at least 0.15, a condition that is not met.



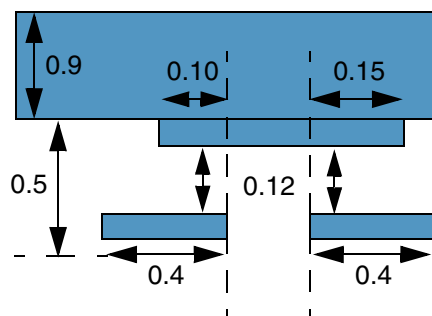
j) FAIL. The neighboring wire is 0.3 long and requires a spacing of at least 0.15 (it need not necessarily have a protrusion).

Virtuoso Technology Data Constraints Reference

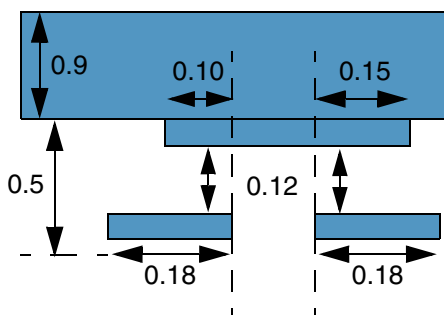
Spacing Constraints (One Layer)



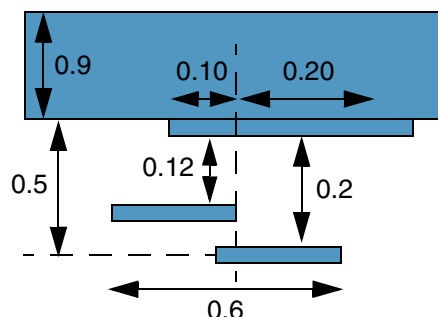
k) The constraint does not apply because the parallel run length of the portion that is within 0.5 of the wide wire is only 0.29 (<0.3).



l) PASS. All triggering conditions are met. The individual projected parallel run lengths of the two protrusions are 0.10 and 0.15 (both less than 0.2); therefore, a spacing of 0.08 is required and is met.



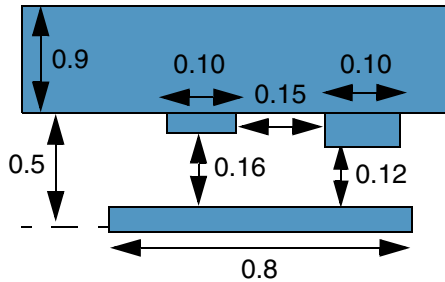
m) The constraint does not apply because the two neighboring wires have individual parallel run lengths of 0.18 (<0.3) with the wide wire.



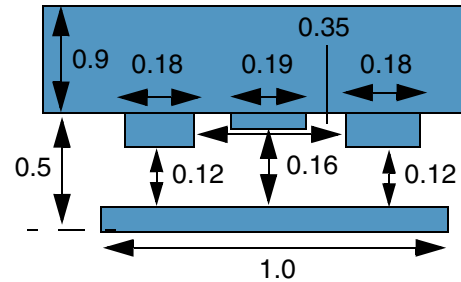
n) PASS. The combined parallel run length of the two neighboring wires is 0.6 (>0.3). The projected parallel run lengths of the two segments of the protrusion are 0.10 and 0.2 (≤ 0.2); therefore, a spacing of 0.08 is required and is met.

Virtuoso Technology Data Constraints Reference

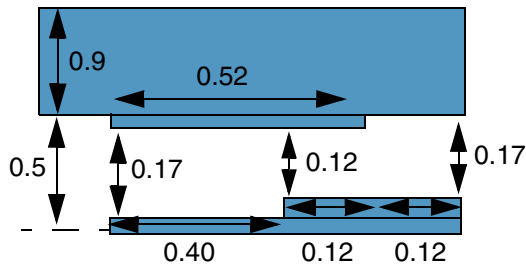
Spacing Constraints (One Layer)



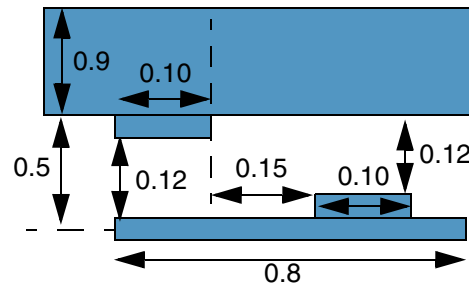
o) PASS. All triggering conditions and the spacing requirement between the wires is met. The spacing between the protrusions does not apply because the protrusion that has a spacing greater than 0.15 is only 0.10 wide (must be greater than 0.2 for the protrusion-to-protrusion spacing condition to apply).



p) FAIL. The left and right protrusions have projected parallel run length less than 0.2 and spacing less than 0.15. Therefore, the distance between the two protrusions must be greater than or equal to 0.7, a condition that is not satisfied. The protrusion in the middle has spacing greater than 0.15, and is, therefore, not considered.



q) PASS. The protrusion segment that is 0.40 (>0.2) wide has a spacing requirement of 0.15, which is met. The protrusion segment that is 0.12 (<0.2) wide has a spacing requirement of 0.8, which is also met. Similarly, the 0.24 wide protrusion on the bottom wire is also broken into two segments with width 0.12 (<0.2) each. Each of these segments has spacing greater than 0.8.



r) FAIL. Both protrusions have projected parallel run length less than 0.2 and spacing less than 0.15. Therefore, the distance between the two protrusions must be greater than or equal to 0.7, a condition that is not satisfied.

minSameNetSpacing (One layer)

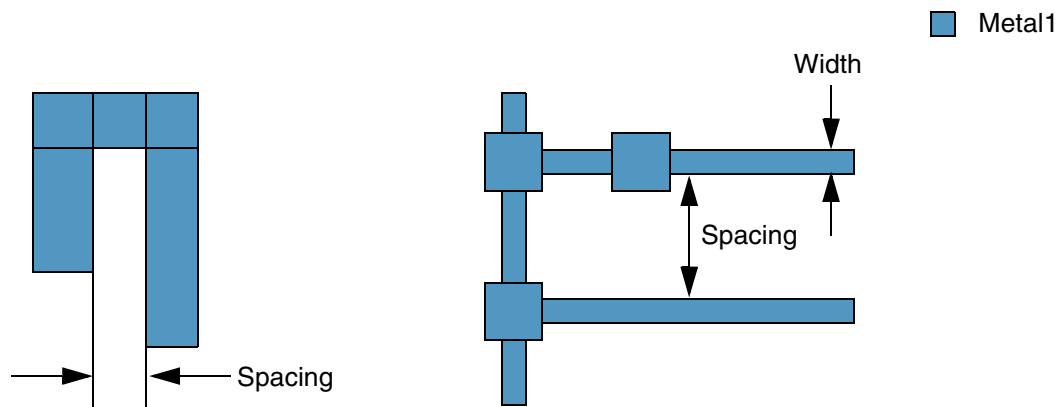
```

spacings (
    ( minSameNetSpacing tx_layer
      f_spacing
    )
) ;spacings

spacingTables (
    ( minSameNetSpacing tx_layer
      (( "width" nil nil )
        [f_default]
      )
      (g_table)
    )
) ;spacingTables
    
```

Specifies the minimum spacing between shapes on the same net (electrically equivalent geometries) on a layer. The spacing between the shapes can be dependent on the width of the shapes.

This constraint is required if same-net spacing is less than the different-net spacing specified by the `minSpacing (One layer)` constraint.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between same-net shapes on a layer must be greater than or equal to this value.
"width" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of a <i>table</i> row is as follows: (<i>f_width f_spacing</i>) where, <i>f_width</i> is the width of the geometry and <i>f_spacing</i> is the minimum spacing required between shapes with width greater than or equal to <i>f_width</i> . Type: A 1-D table specifying floating-point width and spacing values

Parameters

<i>f_default</i>	The spacing value to be used when no table entry applies.

Example

The same-net spacing on Poly1 must be at least 1.0 and the same-net spacing on Metal2 must be at least equal to the value of the technology parameter `minsamenetspace1`.

```
spacings (
  ( minSameNetSpacing "Poly1"
    1.0
  )
  ( minSameNetSpacing "Metal2"
    techParam("minsamenetspace1")
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minSideSpacing (One layer)

```
spacings (
  ( minSideSpacing tx_layer
    ['otherLayer tx_otherLayer
      ['cutClass {f_width | (f_width f_length) | t_name}]
      ['cutWithin g_within]
    ]
    ['shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
      | 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide
    ]
    ['prl f_prl | 'prlRange g_prlRange]
    ['exactAlignedSpacingRange]
    ['sameMask]
    ['cornerEuclidian]
    ['facingEdges]
    ['widthRanges (g_widthRanges)]
    ['lengthRanges (g_lengthRanges)]
    ['otherWidthRanges (g_otherWidthRanges)]
    ['otherLengthRanges (g_otherLengthRanges)]
    ['horizontal | 'vertical]
    ['deltaVoltage f_voltage]
    ['exceptSpacingRanges l_exceptSpacingRanges]
    ['otherFacingEdgeRange g_fRange
      ['facingEdgeHorizontal | 'facingEdgeVertical]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['exceptIntersectLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['widthDiff f_widthDiff]
    ['orthogonalPair]
    ['mask1 | 'mask2 | 'mask3]
    ['widthHorizontal | 'widthVertical]
    f_spacing
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between two shapes on a layer based on which side of the first shape faces which side of the second shape ("other" shape). Sides can be of type short, long, or any.

If the parallel run length between the shapes is less than zero, spacing can optionally be measured corner-to-corner using Euclidean measure type. By default, spacing is measured using Manhattan measure type.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Note: Sides of non-rectangular shapes are of type "any" (that is, they are not considered as short or long).

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shapes must be greater than or equal to this value.

Parameters

`'otherLayer tx_otherLayer`
(Virtuoso Layout Suite EXL and higher tiers) The constraint applies to the specified cut layer that is inside the metal layer.

`'cutClass {f_width | (f_width f_length) | t_name}`
(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when *tx_otherLayer* is of the specified cut class size. The cut class is specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class. If not specified, any cut combination of four cuts can trigger the rule.

`'cutWithin g_cutWithin`
(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if *tx_otherLayer* is within the specified distance. The default value is 0. The distance is measured in the direction orthogonal to the spacing direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
| 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide

The sides between which spacing is measured.

- 'shortSideToShortSide: Spacing is measured between a short side of one shape and a short side of the other shape.
- 'shortSideToLongSide: Spacing is measured between a short side of one shape and a long side of the other shape.
- 'shortSideToAnySide: Spacing is measured between a short side of one shape and any side of the other shape.
- 'longSideToLongSide: Spacing is measured between a long side of one shape and a long side of the other shape.
- 'longSideToAnySide: Spacing is measured between a long side of one shape and any side of the other shape.
- 'anySideToAnySide: Spacing is measured between any side of one shape and any side of the other shape.

'prl f_prl

The constraint applies only if the parallel run length between the two sides is greater than this value.

Type: Boolean

'prlRange g_prlRange

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the parallel run length between the two sides falls in this range.

Type: Floating-point values specifying a range of parallel run lengths that trigger the constraint

'exactAlignedSpacingRange

(Virtuoso Advanced Node for Layout Only) The spacing between the exactly aligned shapes can be in this range as an exception, but only for positive values of the prlRange parameter.

'sameMask

(Virtuoso Advanced Node for Layout Only) The constraint applies only between shapes on the same mask.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

<code>'cornerEuclidian</code>	(Virtuoso Advanced Node for Layout Only) The corner-to-corner spacing (when the parallel run length is less than zero) between shapes is measured using Euclidean measurement type (the spacing halo has rounded corners). The default measurement type is Manhattan. Type: Boolean
<code>'facingEdges</code>	(Virtuoso Advanced Node for Layout Only) The constraint applies only if they face each other and there is no obstruction between edges.
<code>'widthRanges</code> (<i>g_widthRanges</i>)	The constraint applies only if the width of the first shape falls in one of these ranges. Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint
<code>'lengthRanges</code> (<i>g_lengthRanges</i>)	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the length of the first shape falls in one of these ranges. Type: Floating-point values specifying a <u>range</u> of lengths that trigger the constraint
<code>'otherWidthRanges</code> (<i>g_otherWidthRanges</i>)	The constraint applies only if the width of the other shape falls in one of these ranges. Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint
<code>'otherLengthRanges</code> (<i>g_otherLengthRanges</i>)	(Virtuoso Advanced Node for Layout Only) The constraint applies only if the length of the other shape falls in one of these ranges. Type: Floating-point values specifying a <u>range</u> of lengths that trigger the constraint
<code>'horizontal</code> <code>'vertical</code>	

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The direction in which spacing is measured. If direction is not specified, spacing is measured in any direction.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'deltaVoltage f_voltage`

The constraint applies only if the maximum voltage difference between the two shapes is greater than this value.

Delta voltage = $\max(\max1, \max2) - \min(\min1, \min2)$, where $(\max1, \min1)$ denotes the voltage range for the first shape and $(\max2, \min2)$ denotes the voltage range for the second shape.

`'exceptSpacingRanges l_exceptSpacingRanges`

The constraint does not apply if the spacing between the shapes is in this range.

Type: A list of floating-point spacing ranges to which the constraint does not apply

`'otherFacingEdgeRange g_fRange ['facingEdgeHorizontal |
'facingEdgeVertical]`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if one of the two corners has a facing edge within the specified range and direction. You cannot specify a value greater than or equal to the specified range value.

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'exceptIntersectLayers (tx_layer1 tx_layer2 ... tx_layerN)`

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply if the spacing gap is covered by shapes on one of the specified layers.

`'widthDiff f_widthDiff`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the width difference of the shapes is greater than the specified value.

`'orthogonalPair`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to the shapes in orthogonal directions. This parameter is applicable only for shapes with negative parallel run length.

`'mask1 | 'mask2 | 'mask3`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to shapes on the specified mask.

Type: Boolean

`'widthHorizontal | 'widthVertical`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Advanced Node for Layout Only) The direction in which the `widthRanges`, `otherWidthRanges`, `lengthRanges`, and `otherLengthRanges` parameters are measured.

- The `widthRanges` and `otherWidthRanges` parameters would be always measured in this direction, even if it does not correspond to the smaller dimension.
- The `lengthRanges` and `otherLengthRanges` parameters would be measured in the orthogonal direction, even if it does not correspond to the larger dimension.

By default, when this parameter is not used, then the smaller dimension is used for `widthRanges` and `otherWidthRanges` and the larger dimension is used for `lengthRanges` and `otherLengthRanges`.

Type: Boolean

Examples

- Example 1: `minSideSpacing` with `shortSideToShortSide`, `pri`, `widthRanges`, and `otherWidthRanges`
- Example 2: `minSideSpacing` with `shortSideToLongSide` and `deltaVoltage`
- Example 3: `minSideSpacing` with `shortSideToLongSide` and `exceptSpacingRanges`
- Example 4: `minSideSpacing` with `facingEdges`
- Example 5: `minSideSpacing` with `otherFacingEdgeRange`
- Example 6: `minSideSpacing` with `otherLayer`
- Example 7: `minSideSpacing` with `exceptIntersectLayers`
- Example 8: `minSideSpacing` with `widthDiff`
- Example 9: `minSideSpacing` with `orthogonalPair`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

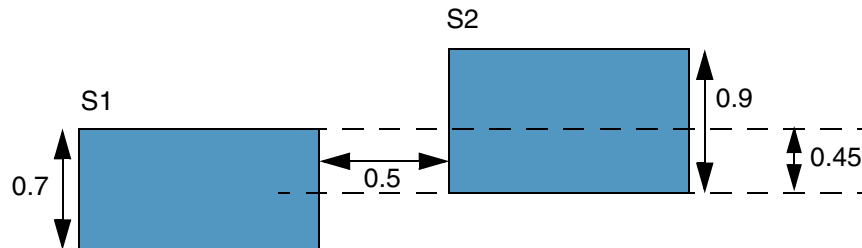
Example 1: *minSideSpacing* with *shortSideToShortSide*, *prl*, *widthRanges*, and *otherWidthRanges*

The spacing between the short sides of shapes S1 and S2 must be greater than or equal to 0.4 if the following conditions are met:

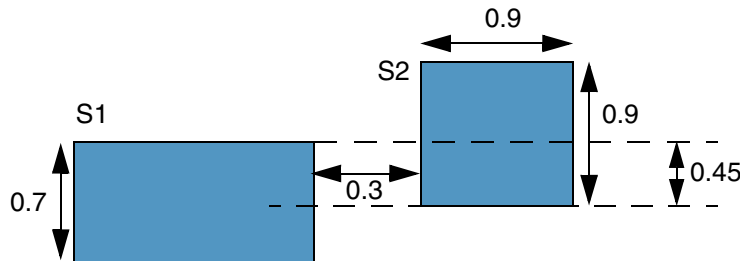
- The width of S1 is greater than or equal to 0.6 and less than or equal to 0.8.
- The width of S2 is greater than or equal to 0.9 and less than or equal to 1.0.
- The parallel run length between the sides is greater than 0.

```
spacings (  
  ( minSideSpacing "Metal1"  
    'shortSideToShortSide  
    'prl 0.0  
    'widthRanges ("[0.6 0.8]")  
    'otherWidthRanges ("[0.9 1.0]")  
    0.4  
  )  
) ;spacings
```

■ Metal1



a) PASS. The widths of both shapes fall in the specified width ranges. The parallel run length between the short sides is greater than zero and the spacing between the short sides is 0.5 (>0.4).



b) The constraint does not apply because the other shape (S2) with both sides equal to 0.9 is a square (its sides are of type any).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

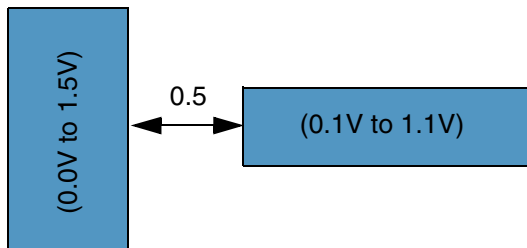
Example 2: *minSideSpacing with shortSideToLongSide and deltaVoltage*

The spacing between a long side and a short side of two adjacent Metal1 shapes must be greater than or equal to 0.6 if the delta voltage between the shapes is greater than 1.5.

```
spacings(  
    ( minSideSpacing "Metal1"  
      'shortSideToLongSide  
      'deltaVoltage 1.5  
      0.6  
    )  
); spacings
```

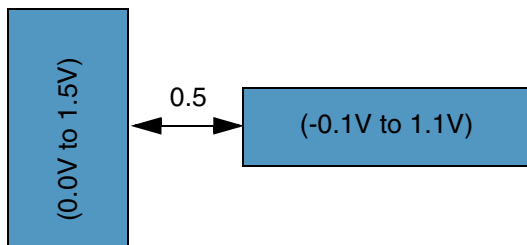
■ Metal1

```
Delta voltage = max(max1, max2) - min(min1, min2)  
               = max(1.5, 1.1) - min(0.0, 0.1)  
               = 1.5 - 0.0 = 1.5
```



a) The constraint does not apply because the voltage difference is only 1.5V (and not greater than 1.5V).

```
Delta voltage = max(max1, max2) - min(min1, min2)  
               = max(1.5, 1.1) - min(0.0, -0.1)  
               = 1.5 - (-0.1) = 1.6
```



b) FAIL. The voltage difference is 1.6V (>1.5V). However, the spacing between the shapes is 0.5 (<0.6).

Virtuoso Technology Data Constraints Reference

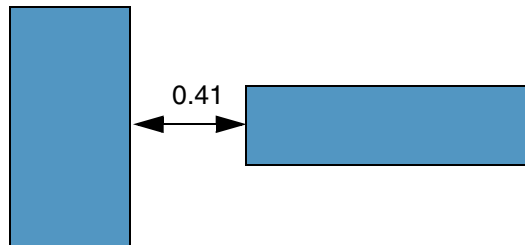
Spacing Constraints (One Layer)

Example 3: minSideSpacing with shortSideToLongSide and exceptSpacingRanges

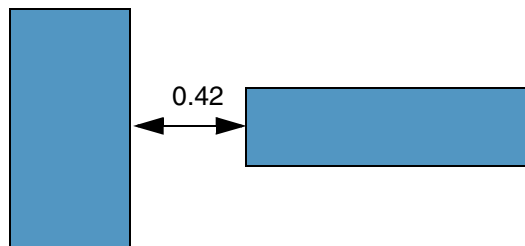
The spacing between a long side and a short side of two adjacent Metal1 shapes must be greater than or equal to 0.6. The constraint does not apply if the spacing is equal to 0.3 or greater than 0.4 and less than 0.42.

```
spacings (  
  ( minSideSpacing "Metal1"  
    'shortSideToLongSide  
    'exceptSpacingRanges (0.3 "(0.4 0.42)")  
    0.6  
  )  
) ;spacings
```

■ Metal1



a) The constraint does not apply because the spacing of 0.41 is in the exception range.



b) FAIL. The spacing of 0.42, which is outside the exception range, is less than 0.6.

Virtuoso Technology Data Constraints Reference

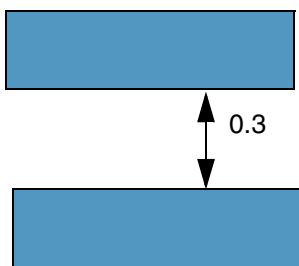
Spacing Constraints (One Layer)

Example 4: minSideSpacing with facingEdges

The constraint applies only if edges are facing and the spacing is 0.4 or greater.

```
spacings(  
  ( minSideSpacing "Metal1"  
    'facingEdges  
    0.4  
  )  
) ;spacings
```

■ Metal1



FAIL: The constraint is applicable because the edges are facing each other, but fails because spacing is less than 4.

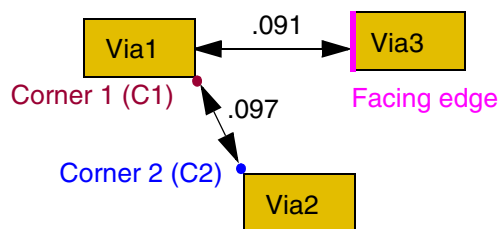
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

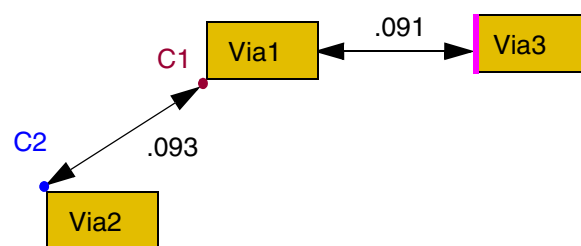
Example 5: *minSideSpacing* with *otherFacingEdgeRange*

The constraint applies only if the edges are facing and the spacing is 0.094 or greater.

```
spacings (  
  ( minSideSpacing "Via1"  
    'facingEdgeHorizontal  
    'cornerEuclidean  
    'otherFacingEdgeRange "<=.108"  
    .094  
  )  
) ;spacings
```



a) The constraint applies because C1 has a facing edge at .091 ($\leq .108$) in the horizontal direction. It passes because the distance between C1 and C2 is .097 ($\geq .094$).



b) The constraint does not apply between C1 and C2 because the corners do not have a facing edge.

Virtuoso Technology Data Constraints Reference

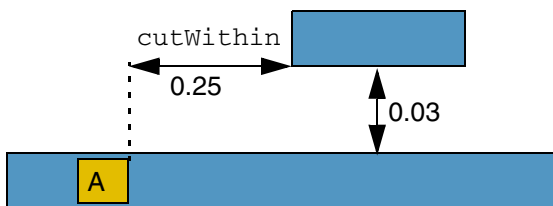
Spacing Constraints (One Layer)

Example 6: *minSideSpacing with otherLayer*

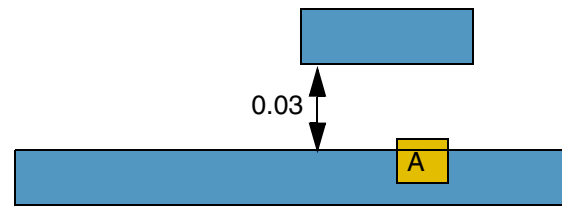
The spacing between M0 and VC must not exceed 0.04 with the given cutClass and cutWithin values.

```
spacings(  
  ( minSideSpacing "M0"  
    'vertical  
    'otherLayer "VC"  
    'cutClass "A"  
    'cutWithin 0.3  
    0.04  
  )  
) ;spacings
```

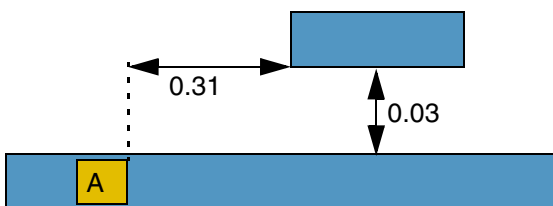
VC
M0



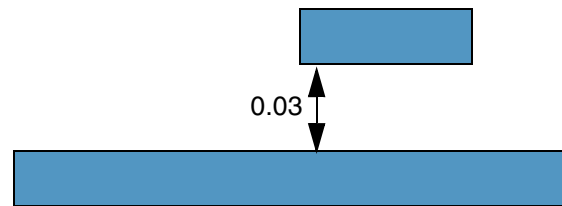
FAIL. The constraint applies because the cut of cutclass A is inside the metal within 0.25, which is less than or equal to 0.3. It fails because the spacing is 0.03, which is less than 0.04.



FAIL. The constraint applies because the cut of cutclass A intersects the metal. It fails because the spacing is 0.03, which is less than 0.04.



PASS. The constraint does not apply and passes because the cut of cutclass A is not inside the metal within 0.3.



PASS. The constraint does not apply because no cutclass A is inside any of the metal.

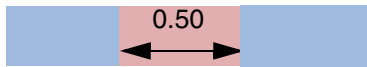
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

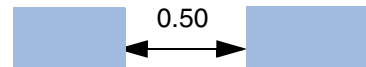
Example 7: minSideSpacing with exceptIntersectLayers

```
spacings (
  ( minSideSpacing "ndiff"
    'shortSideToShortSide
    'exceptIntersectLayers ("layer2")
    0.54
  )
) ;spacings
```

■ ndiff
■ layer2



PASS. The constraint does not apply because the gap is filled with "layer2". As a result, the spacing $0.5 < 0.54$ is not a violation.



FAIL: The constraint applies because the gap is not filled. The constraint fails because $0.5 < 0.54$.

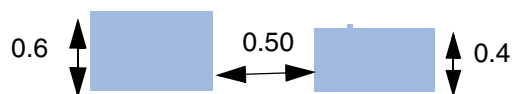
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

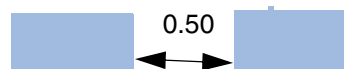
Example 8: minSideSpacing with widthDiff

```
spacings (
  ( minSideSpacing "ndiff"
    'shortSideToShortSide
    'widthDiff 0.12
    0.54
  )
) ;spacings
```

■ ndiff
■ layer2



FAIL. The constraint applies because the width difference is $0.6 - 0.4 = 0.2$ which is > 0.12 , and the spacing of $0.5 < 0.54$ is violated.



PASS. The constraint does not apply because the width difference is < 0.12 .

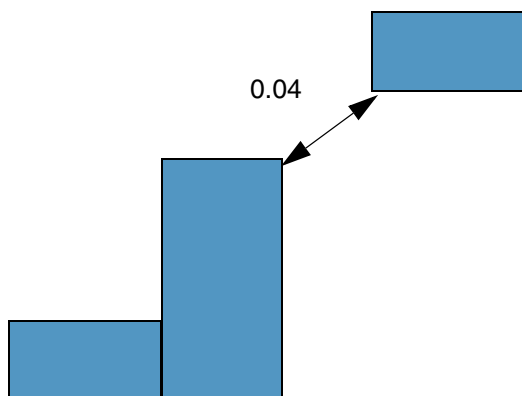
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

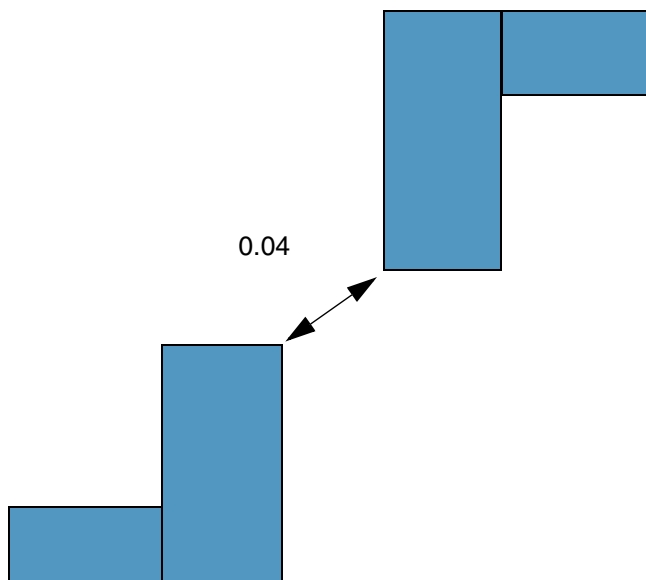
Example 9: minSideSpacing with orthogonalPair

```
spacings (  
  ( minSideSpacing "Metal1"  
    'anySideToAnySide  
    'orthogonalPair  
    'cornerEuclidian  
    'prlRange "<0"  
    0.05  
  )  
) ;spacings
```

■ Metal1



FAIL. The constraint applies because the lower wire is vertical and the upper wire is horizontal and the minimum spacing is violated.



PASS. The constraint does not apply because both wires are in vertical direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minSpacing (One layer)

```
spacings (
  ( minSpacing tx_layer
    ['inLayerDir tx_layer2]
    [{'horizontal' | 'vertical' | 'diagonalOnly'} ['eolWidth f_eolWidth]]
    ['sameNet ['PGNet] | 'sameMetal]
    ['sameMask]
    ['ignoreShapesInRanges ((("width"))(g_ignoreTable))]
    ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    ['outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ]
  ['cornerAllowed]
  ['length f_length]
  ['intersectLayers (tx_layer2 tx_layer3 ... tx_layerN)]
  f_spacing ['manhattan]
)

( minSpacing tx_layer
  ['area f_maxArea]
  f_spacing ['manhattan]
)

) ;spacings

spacingTables (
  ( minSpacing tx_layer
    (( "width" nil nil ["width" | "length" nil nil] )
    ['inLayerDir tx_layer2]
    [{'horizontal' | 'vertical' | 'diagonalOnly'} ['eolWidth f_eolWidth]]
    ['sameNet ['PGNet] | 'sameMetal]
    ['sameMask]
    |
    [{'mask1 | 'mask2 | 'mask3} ['ignoreIntermediateShapes]]
  ]
  ['exceptEolWidth f_eolWidth]
  ['ignoreShapesInRanges ((("width"))(g_ignoreTable))]
  ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
  ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  |
  ['outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
  ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ]
  [f_default]
)
  (g_table) ['manhattan]
)
)
```

Virtuoso Technology Data Constraints Reference

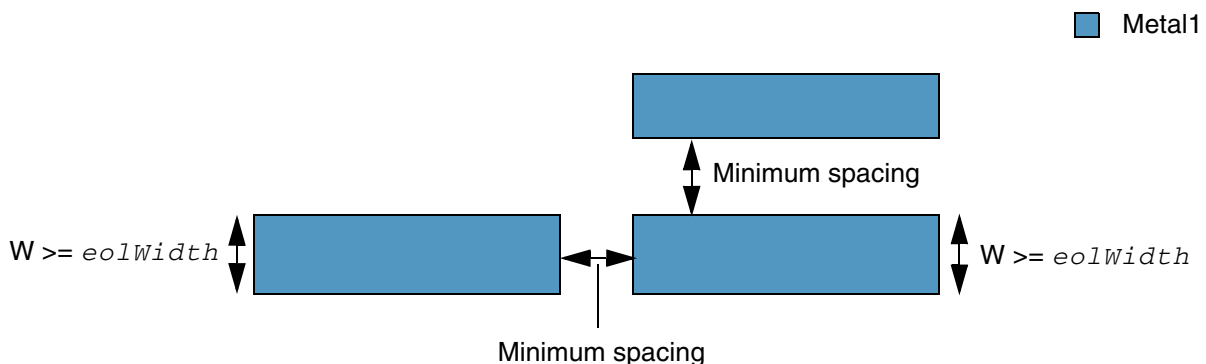
Spacing Constraints (One Layer)

```
( minSpacing tx_layer
  (( "twoWidths" nil nil "length" nil nil )
   ['inLayerDir tx_layer2]
   [{'horizontal | 'vertical | 'diagonalOnly} ['eolWidth f_eolWidth]]
   ['sameNet ['PGNet] | 'sameMetal]
   ['sameMask]
   ['fillConcaveCorner f_fill]
   ['ignoreShapesInRanges (((("width"))(g_ignoreTable)))]
   ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    ['outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
     ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
   ]
   [f_default]
 )
 (g_table) ['manhattan]
 )
) ;spacingTables
```

Specifies the minimum orthogonal spacing required between two adjacent shapes on the specified layer. The required spacing can be dependent on the direction in which the spacing is measured, on the width of the wider of the two shapes, on the width of both the shapes, or on the width of the shapes and the parallel run length between them.

Note: The width of a shape is defined as the smaller of the shape's two dimensions.

If the `'eolWidth` parameter is specified, the constraint applies to edges with width greater than or equal to `eolWidth`.

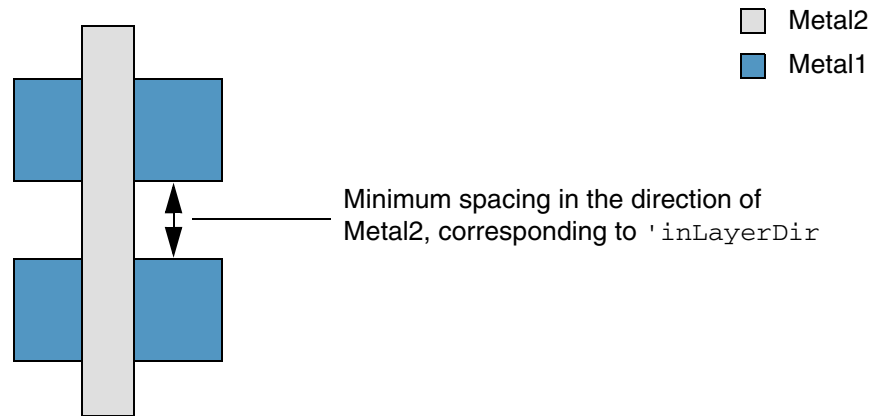


Other optional parameters determine whether the constraint applies to a particular connectivity type or to shapes on power or ground nets. The constraint can also be applied to shapes on the same mask.

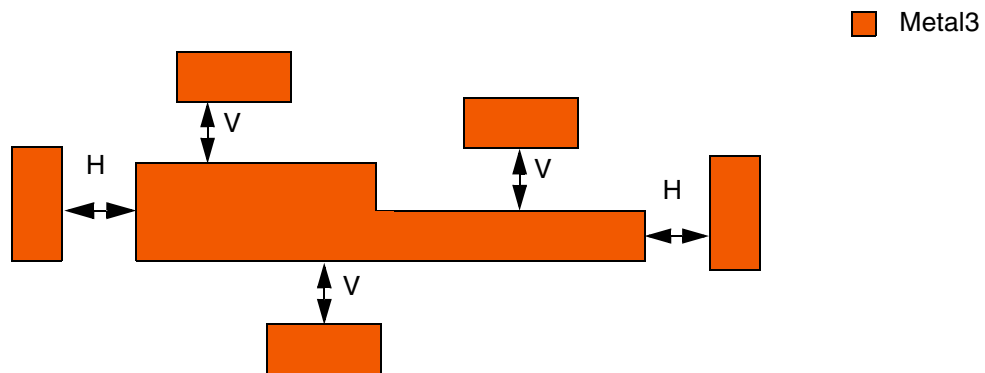
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

If two shapes on the specified layer are overlapped by a single shape on another layer, you can optionally specify this second layer to indicate the direction in which the constraint applies. This is useful if the structures in the design can be used at various rotations. For example, in the figure below, spacing direction for Metal2 is specified as vertical; therefore, spacing between Metal1 shapes is measured in the vertical direction. If this structure is now rotated by 90 degrees, the spacing between Metal1 shapes, after rotation, would be measured horizontally.



In some advanced 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. For example, the preferred routing direction for Metal3 is vertical. Therefore, the spacing measured horizontally (between vertical Metal3 shapes) must meet the regular spacing requirement, and the spacing measured vertically (between horizontal Metal3 shapes) must be greater than the regular spacing.



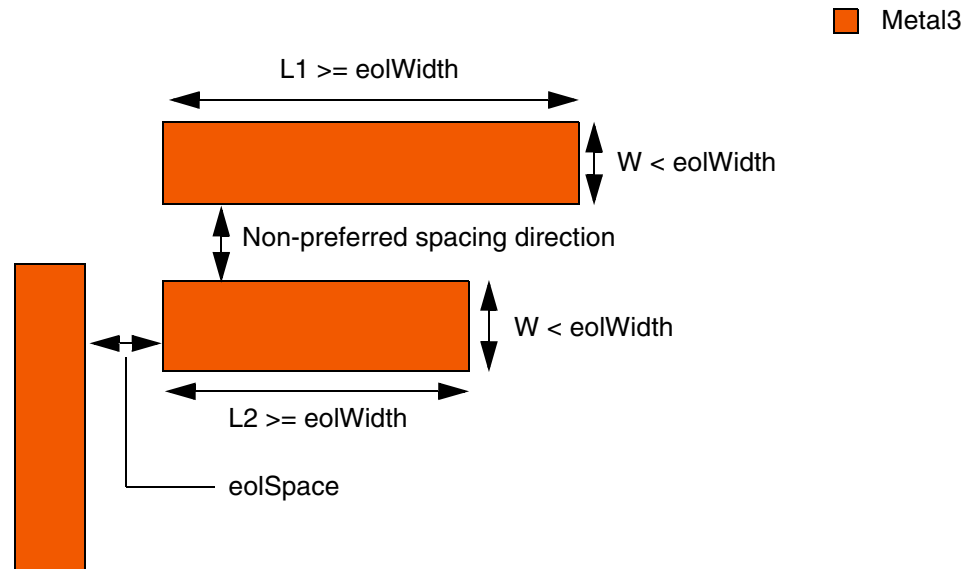
H = Horizontal spacing; V = Vertical spacing

The spacing measured between two end-of-line edges or between an end-of-line edge and a side edge is determined by an end-of-line constraint (for example,

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`minEndOfLineSpacing`), instead of `minSpacing`. The width of an end-of-line edge is less than `eolWidth` and the width of a side edge is greater than or equal to `eolWidth`.



Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>f_spacing</code>	The minimum spacing required between two shapes.
<code>"width" nil nil ["width" "length" nil nil]</code>	This identifies the index for <code>table</code> .
<code>"twoWidths" nil nil "length" nil nil</code>	This identifies the index for <code>table</code> .

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

g_table

The format of a 1-D table is as follows:

```
( f_index f_value  
  ...  
)
```

where, *f_index* is the width of the wider of the two shapes and *f_value* is the minimum spacing that applies when the width is greater than or equal to the corresponding index.

The format of a 2-D table is as follows:

```
( ( f_index1 f_index2 ) f_value  
  ...  
)
```

where,

- *f_index1* and *f_index2* both represent widths (of the two shapes), or *f_index1* is the width of the wider of the two shapes and *f_index2* is the parallel run length between the two shapes.
- *f_value* is the minimum spacing that applies when the width and parallel run length are both greater than or equal to the corresponding indexes.

Note: If *twoWidths* is specified, the lookup table establishes the minimum spacing between a pair of shapes based on the widths of both shapes and the parallel run length between them. Additionally, the widths and the parallel run length must be greater than the corresponding indexes (and not greater than or equal to).

Note: In a spacing table, all values should be explicitly specified. When a value is not specified, the default value *f_default* is used.

Type: A 1-D table specifying floating-point width and spacing values, or a 2-D table specifying floating-point width, parallel run length, and spacing values

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'inLayerDir tx_layer2`

The constraint applies in the direction of this layer.

Type: String (layer name) or Integer (layer number)

`'horizontal | 'vertical | 'any | 'diagonalOnly`

The direction in which the constraint applies.

- `'horizontal`: The constraint applies only in the horizontal direction.
- `'vertical`: The constraint applies only in the vertical direction.
- `'any`: The constraint applies in both horizontal and vertical directions. This is the default value.
- `'diagonalOnly`: (Virtuoso Layout Suite EXL and higher tiers) The constraint applies between a diagonal wire to any wire.

Type: Boolean

`'eolWidth f_eolWidth`

The constraint applies only if the width of an edge is greater than or equal to this value.

This parameter must be used in conjunction with the direction parameter.

`'sameNet ['PGNet] | 'sameMetal`

The connectivity type. If connectivity type is not specified, the constraint applies to any connectivity type, including any two shapes on the same layer with no common metal or net.

- `'sameNet`: The constraint applies only to the shapes on the same net.
- `'PGNet`: The constraint applies only to the shapes on the same power or ground net.
- `'sameMetal`: The constraint applies only to the shapes that form a contiguous same-metal shape.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'sameMask (Virtuoso Advanced Node for Layout Standard) The constraint applies only to shapes on the same mask.

'mask1 | 'mask2 | 'mask3

(Virtuoso Advanced Node for Layout Only) The constraint applies only to the shapes on this mask.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'ignoreIntermediateShapes`

(Virtuoso Advanced Node for Layout Only) The constraint applies between two shapes if they are on the specified mask, irrespective of whether a shape, gray or on any other mask, is present between them.

Type: Boolean

`'area f_maxArea`

The constraint applies between a shape with area less than this value and all other shapes on the specified layer.

No other parameters are permitted in combination with this parameter.

`'ignoreShapesInRanges ((("width")) (g_ignoreTable))`

(Virtuoso Advanced Node for Layout Standard) Spacing between two shapes on a layer is ignored if the width of the wider shape is greater than or equal to the specified width and the spacing between the two shapes is within the ignore range for that width.

The range of spacing values to be ignored must be less than the minimum spacing for the specified width.

Each row of *ignoreTable* has the following format:

f_width (*g_range*)

where,

- *f_width* specifies the width.
- *g_range* specifies the range of spacing values that are ignored.

Type: A 1-D table specifying floating-point width and spacing values

`'exceptEolWidth f_eolWidth`

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply to edges with width less than this value.

`'fillConcaveCorner f_fill`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Advanced Node for Layout Only) Before spacing is checked between a concave corner and a shape, the concave corner is filled with a triangular shape whose sides measured along the concave edges are equal to this value. Moreover, the spacing between this concave fill and a shape must be greater than or equal to the smallest spacing value specified in the table, provided this minimum spacing applies to all related shapes.

```
'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer3 layer4 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

f_default

The spacing value to be used when no table entry applies or is specified (using *g_table*). Ensure that the default spacing value is sufficiently large to represent all missing spacing values in the table. Ideally, it should be greater than or equal to the largest specified spacing in the table.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'cornerAllowed	(Virtuoso Advanced Node for Layout Only) The constraint does not apply if the corners of the two shapes touch.
'length <i>f_length</i>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to shapes with length less than this value.
'intersectLayers (<i>tx_layer2 tx_layer3 ... tx_layerN</i>)	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if the spacing gap is covered by shapes on one of the specified layers.
'manhattan	The constraint uses Manhattan distance, which allows a larger spacing at the corners. By default, the constraint uses Euclidean measurement. Type: Boolean

Examples

- [Example 1: minSpacing \(fixed value\)](#)
- [Example 2: minSpacing with width \(1-D table\)](#)
- [Example 3: minSpacing with width and width \(2-D table\)](#)
- [Example 4: minSpacing with width and length \(2-D table\)](#)
- [Example 5: minSpacing with width and length and default spacing \(2-D Table\)](#)
- [Example 6: minSpacing with twoWidths and length \(2-D table\)](#)
- [Example 7: minSpacing with ignoreShapesInRanges](#)
- [Example 8: minSpacing with sameMask](#)
- [Example 9: minSpacing with fillConcaveCorner](#)
- [Example 10: minSpacing with area](#)
- [Example 11: minSpacing with mask1 and ignoreIntermediateShapes](#)
- [Example 12: minSpacing with intersectLayers](#)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 1: minSpacing (fixed value)

The minimum spacing on layer Poly1 is 0.5 and the minimum spacing on layer Metal2 is equal to the value of the technology parameter `minspacing1`.

```
spacings (
  ( minSpacing "Poly1" 0.5 )
  ( minSpacing "Metal2" techParam("minspacing1") )
) ;spacings
```

Example 2: minSpacing with width (1-D table)

The lookup table establishes the minimum spacing for a shape on Metal1 based on the width of the shape. For example, if the width of a shape is greater than or equal to 0.0005, but less than 0.5605, the minimum spacing is 0.12; if the width of a shape is greater than or equal to 0.5605, but less than 1.5005, the minimum spacing is 0.18; and so on.

```
spacingTables (
  ( minSpacing "Metal1"
    (( "width" nil nil ))
    (
      0.0005 0.12
      0.5605 0.18
      1.5005 0.50
      3.0005 0.90
      7.5005 2.50
      ...
    )
  )
) ;spacingTables
```

Example 3: minSpacing with width and width (2-D table)

The lookup table establishes the minimum spacing for a shape on Metal2 based on the widths of the two shapes. If the width of one shape is greater than or equal to 0.0005 and the width of another shape is greater than or equal to 0.0001, the minimum spacing is 0.10; if the width of one shape is greater than or equal to 0.0005 and the width of another shape is greater than or equal to 0.0010, the minimum spacing is 0.12; and so on.

```
spacingTables (
  ( minSpacing "Metal2"
    (( "width" nil nil "width" nil nil ))
    (
      (0.0005 0.0001) 0.10
      (0.0005 0.0010) 0.12
      (0.0005 0.0020) 0.14
      (0.0005 0.0030) 0.16
      ...
    )
  )
) ;spacingTables
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 4: minSpacing with width and length (2-D table)

The lookup table establishes the minimum spacing for Metal3 based on the width of the wider shape and the parallel run length (prl) between the two shapes. If the width and prl are both greater than or equal to 0.0005, the minimum spacing is 0.12; if the width is greater than or equal to 0.0005 and the prl is greater than or equal to 0.5605, the minimum spacing is 0.18; and so on.

However, if you index into the table with a width less than the first width index, the first width index is used. Similarly, if you index into the table with a prl less than the smallest prl index (length), the constraint value for the smallest prl index is returned.

```
spacingTables (
  ( minSpacing "Metal3"
    (( "width" nil nil "length" nil nil ))
    (
      (0.0005 0.0005) 0.12
      (0.0005 0.5605) 0.18
      (0.0005 1.5005) 0.50
      (0.0005 3.0005) 0.90
      (0.0005 7.5005) 2.50
      (0.1805 0.0005) 0.18
      (0.1805 0.5605) 0.18
      (0.1805 1.5005) 0.50
      (0.1805 3.0005) 0.90
      (0.1805 7.5005) 2.50
      (1.5005 0.0005) 0.50
      (1.5005 0.5605) 0.50
      (1.5005 1.5005) 0.50
      (1.5005 3.0005) 0.90
      (1.5005 7.5005) 2.50
      (3.0005 0.0005) 0.90
      (3.0005 0.5605) 0.90
      (3.0005 1.5005) 0.90
      (3.0005 3.0005) 0.90
      (3.0005 7.5005) 2.50
      (4.5005 0.0005) 1.50
      (4.5005 0.5605) 1.50
      (4.5005 1.5005) 1.50
      (4.5005 3.0005) 1.50
      (4.5005 7.5005) 2.50
      (7.5005 0.0005) 2.50
      (7.5005 0.5605) 2.50
      (7.5005 1.5005) 2.50
      (7.5005 3.0005) 2.50
      (7.5005 7.5005) 2.50
    )
  )
) ;spacingTables
```

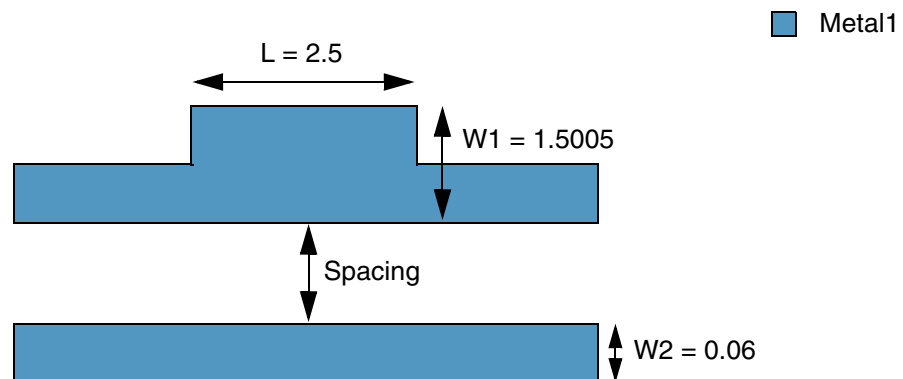
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The following table is a representation of the example above.

Spacing	length >=	0.0005	0.5605	1.5005	3.0005	7.5005
width >=						
0.0005		0.12	0.18	0.50	0.90	2.50
0.1805		0.18	0.18	0.50	0.90	2.50
1.5005		0.50	0.50	0.50	0.90	2.50
3.0005		0.90	0.90	0.90	0.90	2.50
4.5005		1.50	1.50	1.50	1.50	2.50
7.5005		2.50	2.50	2.50	2.50	2.50

In the figure below, the width of the wider of the two shapes (W1) is 1.5005 and the parallel run length (L) between the two shapes is 2.5.



A comparison of W1 and L with the two sets of indexes provides a minimum spacing value of 0.5 — the minimum spacing value found at the intersection of row 3 (index = 1.5005) and column 3 (index = 1.5005).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 5: minSpacing with width and length and default spacing (2-D Table)

The lookup table establishes the minimum spacing for Metal1 based on the width of the wider shape and the parallel run length between the two shapes. The default value of 0.9 is used when a table entry is not specified, say, for (1.0 1.0), as shown below. The effective table entry for (1.0 1.0) in such a case would be 0.9.

```
spacingTables (
  ( minSpacing "Metal1"
    (( "width" nil nil "length" nil nil ) 0.9)
    (
      (0.1 0.5) 0.3
      (0.1 1.0) 0.3
      (0.1 2.0) 0.3
      (1.0 0.5) 0.3
      (1.0 2.0) 0.6
      (2.0 0.5) 0.3
      (2.0 1.0) 0.6
      (2.0 2.0) 1.6
    )
  )
) ;spacingTables
```

The following table is a representation of the example above.

Spacing	length >=	0.5	1.0	2.0
width >=				
0.1		0.3	0.3	0.3
1.0		0.3	?	0.6
2.0		0.3	0.6	1.6

The default value of 0.9 is used in place of the missing value indicated by the ? symbol in the table above.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 6: minSpacing with twoWidths and length (2-D table)

The lookup table establishes the minimum spacing for Metal1 based on the widths of both shapes and the parallel run length between them. A default minimum spacing value of 0.15 applies when no table entry applies.

```
spacingTables(  
  ( minSpacing "Metal1"  
    (( "twoWidths" nil nil "length" nil nil ) 0.15 )  
    (  
      (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  
  
      (1.50 none)  0.50  
      (1.50 0.0 )  0.50  
      (1.50 1.5 )  0.60  
      (1.50 3.0 )  1.00  
  
      (3.00 none)  1.00  
      (3.00 0.0 )  1.00  
      (3.00 1.5 )  1.00  
      (3.00 3.0 )  1.20  
    )  
  ) 'ref "Metal4.Space.1, Metal4.S.2, Metal4.S.3, Metal4.S.4"  
) ;spacingTables
```

Note: The spacing table for `twoWidths` is symmetric, that is, if there are four rows corresponding to one width value, there must be four rows corresponding to each width value. Additionally, duplicate width-length pairs corresponding to different spacing values are allowed.

The following table is a representation of the example above.

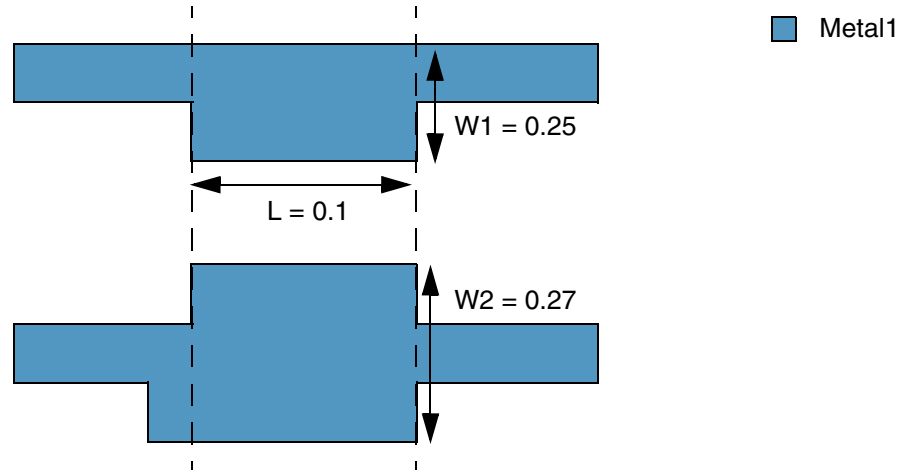
Spacing	length >	none	0.00	1.50	3.00
width >					
0.00		0.15	0.20	0.50	1.00
0.25		0.20	0.25	0.50	1.00
1.50		0.50	0.50	0.60	1.00
3.00		1.00	1.00	1.00	1.20

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Scenario 1

In the figure below, the width of the first shape (W1) is 0.25 and the width of the second shape (W2) is 0.27. The parallel run length (L) between the two shapes is 0.1.



The table is looked up twice to locate the applicable spacing value. During the first pass, W1 and L are looked up horizontally, and, during the second pass, W2 and L are looked up vertically. During each pass, the last width and length index pair that satisfies both "W>width" and "L>length" is located in the table. For example,

- W1 is equal to 0.25. The width index for which $W1 > \text{width}$ is 0.00. The value of the corresponding length index is none, which satisfies $L > \text{length}$.
- W2 is equal to 0.27. The width index for which $W2 > \text{width}$ is 0.25. The value of the corresponding length index is 0.00, which satisfies $L > \text{length}$.

The required spacing lies at the intersection of the row and column corresponding to the two widths and the parallel run length, as shown below:

		width >	0.00	0.25	1.50	3.00
		length >	none	0.00	1.50	3.00
width >	length >					
0.00	none	0.15	0.20	0.50	1.00	W1/L
0.25	0.00	0.20	0.25	0.50	1.00	
1.50	1.50	0.50	0.50	0.60	1.00	
3.00	3.00	1.00	1.00	1.00	1.20	
			W2/L			

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Because the table is symmetrical, the same spacing value (0.20) is obtained if W1 is looked up vertically and W2 is looked up horizontally, as shown below:

		width >	0.00	0.25	1.50	3.00
		length >	none	0.00	1.50	3.00
width >	length >					
0.00	none		0.15	0.20	0.50	1.00
0.25	0.00		0.20	0.25	0.50	1.00 <i>W2/L</i>
1.50	1.50		0.50	0.50	0.60	1.00
3.00	3.00		1.00	1.00	1.00	1.20
			<i>W1/L</i>			

Scenario 2

Next, assume that the width of the first shape (W1) is 0.30 and the width of the second shape (W2) is 1.60. The parallel run length (L) between the two shapes is 1.40.

Again, the table is looked up twice to locate the applicable spacing value. During the first pass, W1 and L are looked up horizontally, and, during the second pass, W2 and L are looked up vertically. During each pass, the last `width` and `length` index pair that satisfies both "`W>width`" and "`L>length`" is located in the table. For example,

- W1 is equal to 0.30. The `width` index for which `W1>width` is 0.25. The value of the corresponding `length` index is 0.00, which satisfies `L>length`.
- W2 is equal to 1.60. The `width` index for which `W2>width` is 1.50. However, the value of the corresponding `length` index is 1.50, which does not satisfy `L>length`. Therefore, `width` index 0.25 is considered (because its corresponding `length` index of 0.00 satisfies `L>length`).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The required spacing lies at the intersection of the row and column corresponding to the two widths and the parallel run length, as shown below:

		width >	0.00	0.25	1.50	3.00	
		length >	none	0.00	1.50	3.00	
width >	length >						
0.00	none		0.15	0.20	0.50	1.00	
0.25	0.00		0.20	0.25	0.50	1.00	<i>W1/L</i>
1.50	1.50		0.50	0.50	0.60	1.00	
3.00	3.00		1.00	1.00	1.00	1.20	
				<i>W2/L</i>			

Because the table is symmetrical, the same spacing value (0.25) is obtained if W1 is looked up vertically and W2 is looked up horizontally, as shown below:

		width >	0.00	0.25	1.50	3.00	
		length >	none	0.00	1.50	3.00	
width >	length >						
0.00	none		0.15	0.20	0.50	1.00	
0.25	0.00		0.20	0.25	0.50	1.00	<i>W2/L</i>
1.50	1.50		0.50	0.50	0.60	1.00	
3.00	3.00		1.00	1.00	1.00	1.20	
				<i>W1/L</i>			

Virtuoso Technology Data Constraints Reference

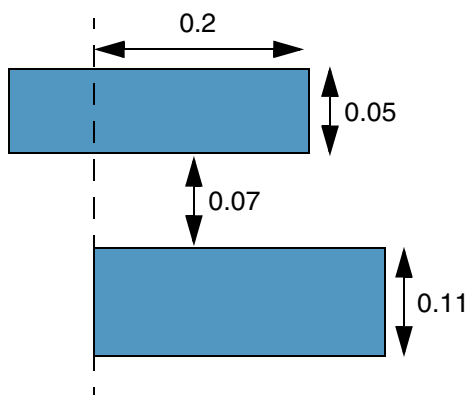
Spacing Constraints (One Layer)

Example 7: minSpacing with ignoreShapesInRanges

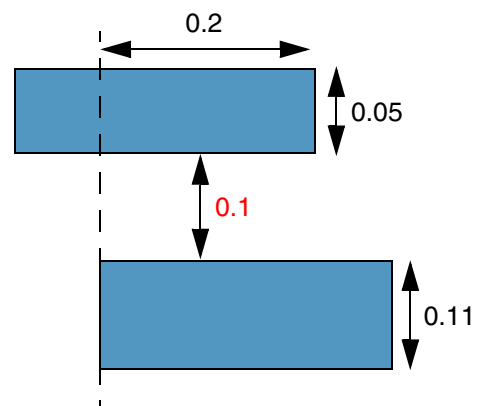
The spacing requirement can vary based on the width of adjacent shapes and the parallel run length between them. For example, the spacing between shapes must be at least 0.14 if the width of the wider of the two shapes is greater than or equal to 0.1 and the parallel run length between the shapes is greater than or equal to 0.08. The constraint does not apply between two shapes if the width of the wider shape is greater than or equal to 0.1 and the spacing between the two shapes is greater than or equal to 0.07 and less than 0.1.

```
spacingTables(  
  ( minSpacing "Metal1"  
    (( "width" nil nil "length" nil nil ))  
    'ignoreShapesInRanges ((("width")) (0.1 ("[0.07 0.10)")))  
  )  
  (  
    (0.0 -0.041) 0.05  
    (0.0 -0.040) 0.06  
    (0.0 0.080) 0.06  
    (0.1 -0.041) 0.05  
    (0.1 -0.040) 0.06  
    (0.1 0.080) 0.14  
  )  
)  
) ;spacingTables
```

■ Metal1
□ Search window



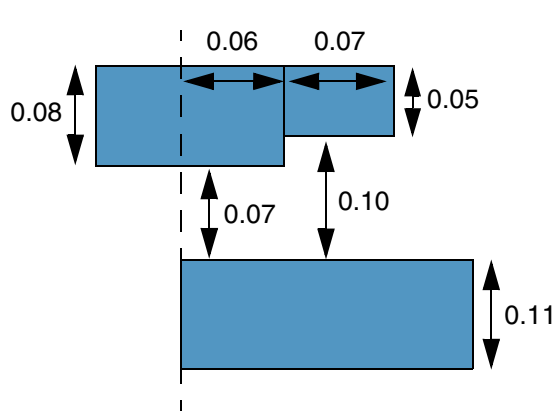
a) The constraint does not apply because the spacing between the two shapes is 0.07, a value that falls in the exempted spacing range.



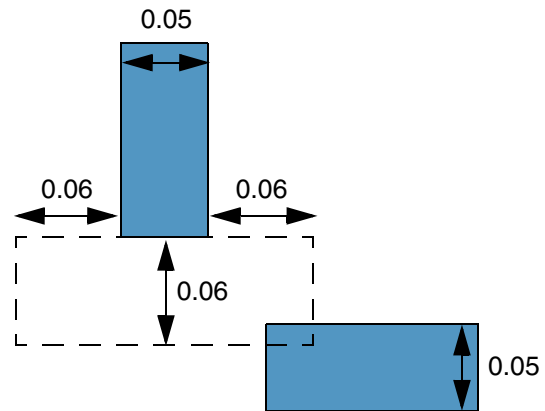
b) FAIL. The spacing between the two shapes lies outside the exemption range and is also less than 0.14 — the spacing required when the width of the wider of the two shapes is 0.11 and the parallel run length between the shapes is greater than or equal to 0.08.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) The constraint does not apply. A spacing of 0.07 falls in the exempted spacing range. The parallel run length of the shape outside the exemption range is 0.07 (< 0.08) and it does not trigger the constraint.



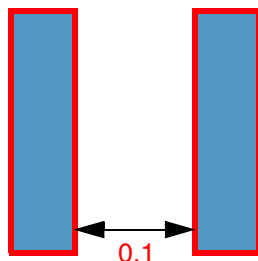
d) FAIL. The required spacing is 0.06 (width ≥ 0 and parallel run length ≥ -0.04). However, the neighboring shape is within a distance of 0.06. (To satisfy the spacing requirement of 0.06, it must be outside the search window.)

Example 8: minSpacing with sameMask

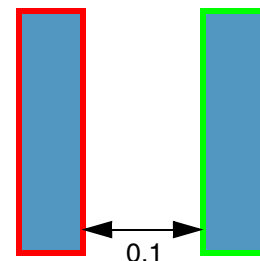
The minimum spacing between two adjacent shapes must be 0.15 if the two shapes are on the same mask, and 0.1 otherwise.

```
spacings (
  ( minSpacing "Metal1"
    0.1
  )
  ( minSpacing "Metal1"
    'sameMask
    0.15
  )
) ;spacings
```

■ Metal1
 ■ mask1
 ■ mask2



a) FAIL. The shapes are on the same mask. However, the spacing between them is 0.1 (< 0.15).



b) PASS. The shapes are on different masks, and therefore, require a minimum spacing of 0.1.

Virtuoso Technology Data Constraints Reference

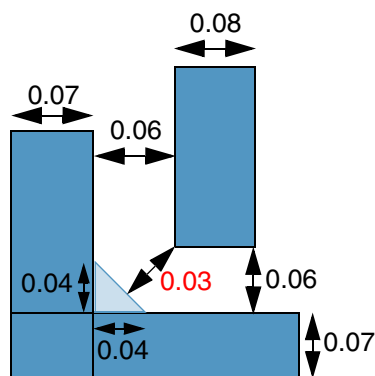
Spacing Constraints (One Layer)

Example 9: minSpacing with fillConcaveCorner

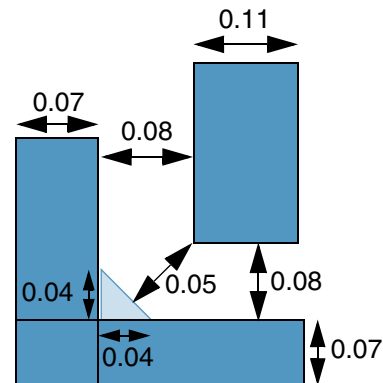
All concave corners must be filled by a triangle whose sides along the concave edges are equal to 0.04 before minimum spacing is checked. The minimum spacing requirement between shapes is 0.06, which also applies to the concave-corner fill. The spacing requirement can vary based on the width of adjacent shapes and is determined by looking up the spacing table. However, this non-minimum spacing requirement does not apply to the concave fill.

```
spacingTables (
  ( minSpacing "Metal1"
    (( "twoWidths" nil nil "length" nil nil )
      'fillConcaveCorner 0.04
    )
  )
  (
    (0.0 0.0) 0.06
    (0.0 0.1) 0.08
    (0.1 0.0) 0.08
    (0.1 0.1) 0.09
  )
) ;spacingTables
```

■ Metal1



a) FAIL. The spacing between adjacent shapes is equal to 0.06, which is the required minimum spacing. However, the spacing between the concave-corner fill and the adjacent shape is 0.03, which is less than the required minimum spacing of 0.06.



b) PASS. The spacing between a shape that is 0.11 wide and the adjacent shapes is 0.08, and this spacing is greater than 0.06, the smallest spacing value specified in the table. Therefore, the spacing requirement of 0.08 does not apply to the concave-corner fill.

Virtuoso Technology Data Constraints Reference

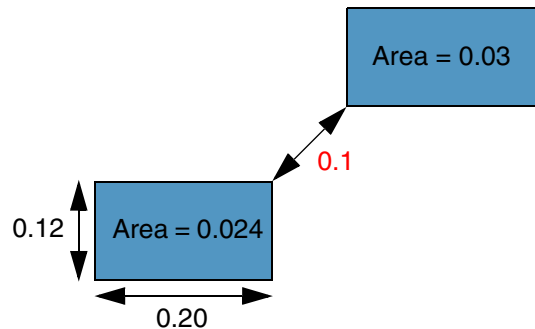
Spacing Constraints (One Layer)

Example 10: minSpacing with area

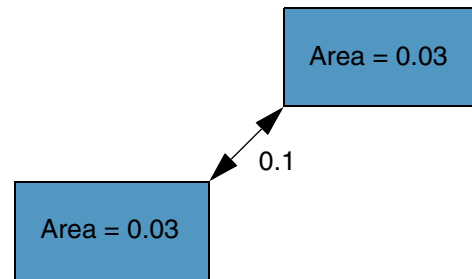
The minimum spacing between two adjacent shapes must be 0.15 if at least one of the shapes has area less than 0.03.

```
spacings (  
  ( minSpacing "Metal1"  
    'area 0.03  
    0.15  
  )  
) ;spacings
```

■ Metal1



a) FAIL. The area of one of the shapes is 0.024 (< 0.03), but its distance from an adjacent shape is 0.1 (< 0.15).



b) The constraint does not apply because the area of both adjacent shapes is 0.03.

Virtuoso Technology Data Constraints Reference

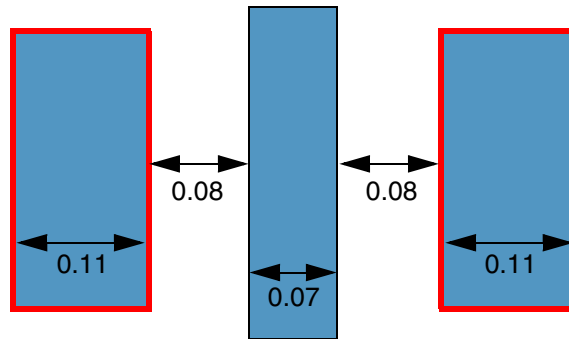
Spacing Constraints (One Layer)

Example 11: *minSpacing with mask1 and ignoreIntermediateShapes*

The spacing between two Metal1 shapes on mask1 must be greater than or equal to 0.3 if both shapes have width greater than or equal to 0.1. If a shape, gray or on any other mask, is present between them, it is ignored.

```
spacingTables (
  ( minSpacing "Metal1"
    (( "width" nil nil "width" nil nil )
      'mask1 'ignoreIntermediateShapes
    )
    (
      (0.0 0.0) 0.04
      (0.0 0.1) 0.06
      (0.1 0.0) 0.06
      (0.1 0.1) 0.30
    )
  )
) ;spacingTables
```

■ Metal1
□ mask1



FAIL. The width of both mask1 shapes is 0.11 (>0.1). Therefore, the spacing between them must be at least 0.3. However, the spacing between them is only 0.23 ($0.08+0.07+0.08$).

Example 12: *minSpacing with intersectLayers*

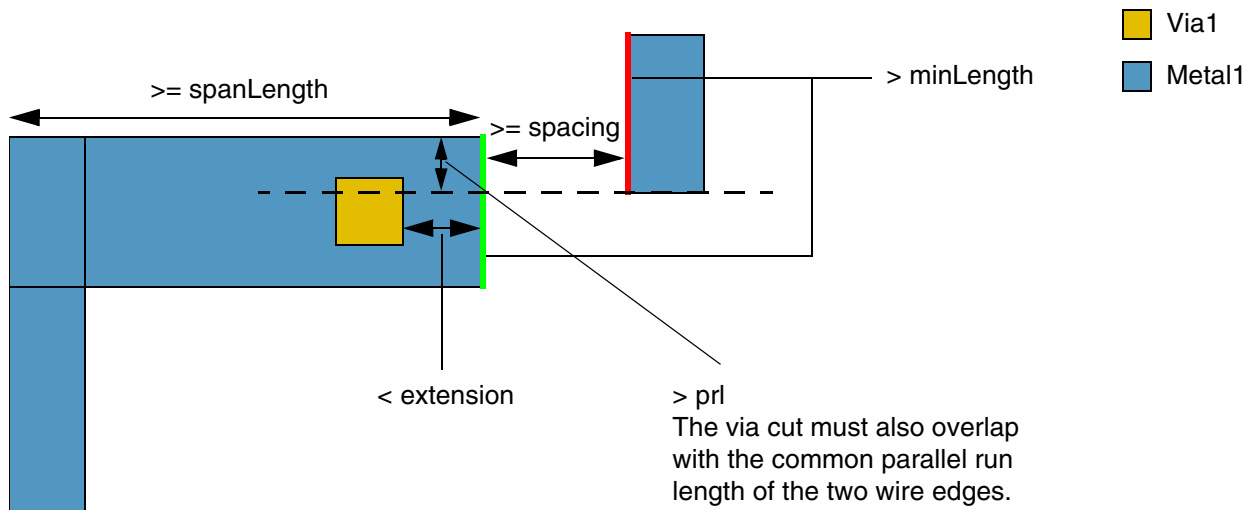
The constraint applies only if the spacing between the shapes on Implant1 is filled by shapes on either Implant2 or Implant3.

```
spacings (
  (minSpacing "Implant1"
    'intersectLayers ("Implant2" "Implant3")
    0.05
  )
) ;spacings
```

minSpanLengthExtensionSpacing

```
orderedSpacings (
    ( minSpanLengthExtensionSpacing tx_metalLayer tx_cutLayer
      'spanLength f_spanLength
      'minLength f_minLength
      'prl f_prl
      'edgeExtension f_extension
      f_spacing
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Only) Defines the spacing between a metal edge that encloses a via cut and a neighboring metal edge with which it shares the required parallel run length. The metal edge that encloses the via cut must satisfy the given *spanLength* and *extension* criteria. Additionally, both metal edges must satisfy the specified *minLength* criterion.



Values

<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

f_spacing The spacing between the metal edges must be greater than or equal to this value.

Parameters

`'spanLength f_spanLength`

The metal edge that encloses a via cut must have span length greater than or equal to this value.

`'minLength f_minLength`

Both metal edges must have length greater than this value.

`'prl f_prl`

The parallel run length between the two metal edges must be greater than this value. The via cut must also overlap with the common parallel run length of the two metal edges.

`'edgeExtension f_extension`

The via cut must have an extension less than this value from the metal edge with span length greater than or equal to *spanLength*.

Example

The spacing between two neighboring Metal1 edges, one of which encloses a Via1 cut, must be greater than or equal to 0.07 if the following conditions are satisfied:

- Both metal edges are greater than 0.05 in length.
- The metal edge that encloses the via cut has span length greater than or equal to 0.1 and an extension of less than 0.01.
- The parallel run length between the two metal edges is greater than 0.03.

```
orderedSpacings (
  ( minSpanLengthExtensionSpacing "Metal1" "Via1"
    'spanLength 0.1
    'minLength 0.05
    'prl 0.03
    'edgeExtension 0.01
    0.07
  )
) ;orderedSpacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

minSpanLengthSpacing

```
spacingTables(  
  ( minSpanLengthSpacing tx_layer  
    (( "prl" nil nil "spanLength" nil nil )  
      ['horizontal | 'vertical | 'any]  
      ['exceptJogLength f_length  
        ['useEdgeLength] ['treatLAsJog]  
        {'horizontalJog | 'verticalJog}  
        ['jogWidth (f_jogWidth f_wireWidth)]  
      ]  
      ['eolWidth f_eolWidth  
        [endOfLineExemptionRanges (g_endOfLineExemptionRanges)]  
      ]  
      ['exactSpacingTable g_exactSpacingTable]  
      ['spacingToMinSpanTable g_spacingToMinSpanTable]  
      ['exactSelfSpacingTable g_exactSelfSpacingTable]  
      ['eolSpacingTable g_eolSpacingTable]  
      ['sameMask | 'diffMask]  
      ['minSpanSpacingRanges g_minSpanTable  
        ['minSpanSpacingRangesPr1 g_minSpanPr1Table]  
      ]  
      ['minSpanForExceptEolWidth f_minSpanEolWidth]  
      ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]  
        |  
        ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]  
      ]  
      ['eolExceptionNoPr1]  
    )  
    (g_table)  
  )  
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) Specifies the spacing between two shapes based on their spans and the parallel run length between them. Optionally, the spacing can apply in the specified direction or to shapes on the same mask. Z- and L-shaped jogs are exempt from the spacing requirement if they meet all the specified criteria.

Additionally, a few exact spacings and a spacing range can be specified for a set of spans, typically the two smallest allowed widths, if the parallel run length between the shapes is non-zero.

Note: Span is the width of a shape measured in the spacing direction and is not always the smallest dimension of the shape. End-of-line width is measured in the direction perpendicular to the span and spacing direction.

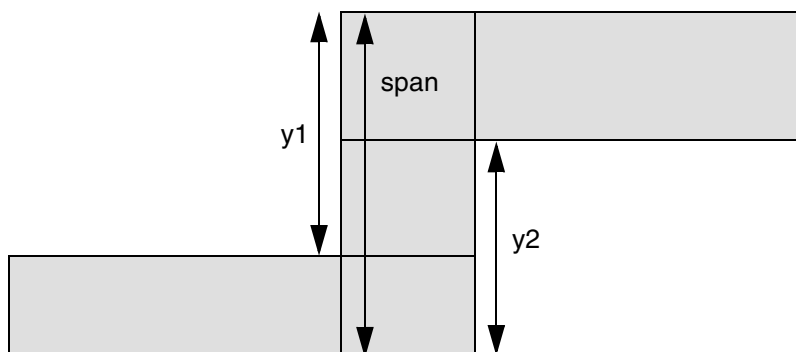
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Optionally, you can use the `'useEdgeLength` parameter to specify how the length in `'exceptJogLength` should be measured.

Preferred routing direction: Horizontal

☐ Metal2



In the figure above, both `y1` and `y2` must be less than `length` for the "Z" shape to qualify for the `'exceptJogLength` exemption. If `'useEdgeLength` is not specified, the `span` (length of the jog) must be less than `length`.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"prl" nil nil "spanLength" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of the table is as follows: (<i>f_prl</i> <i>f_spanLength</i>) <i>f_spacing</i> where, ■ <i>f_prl</i> applies if the parallel run length between the two shapes is greater than or equal to this value. ■ <i>f_spanLength</i> applies if the larger of the two spans is greater than or equal to this value. ■ <i>f_spacing</i> is the spacing required between the two shapes. Type: A 2-D table specifying floating-point parallel run length, spans, and spacing values

Parameters

'horizontal' 'vertical' 'any'	The direction in which the constraint applies. ■ 'horizontal': The constraint applies only in the horizontal direction. ■ 'vertical': The constraint applies only in the vertical direction. ■ 'any': The constraint applies in both horizontal and vertical directions. This is the default value. Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'exceptJogLength f_length`

The constraint does not apply between a Z-shaped jog and a neighboring wire with end-of-line width less than or equal to *wireWidth* and direction perpendicular to the direction in which the length of the jog is measured if the following conditions are satisfied:

- The length of the jog is less than this value.
- The width of the jog is less than or equal to *jogWidth*.
- The width of the adjacent wires is less than or equal to *wireWidth* and their projected parallel run length in the direction of the jog is less than zero.

`'treatLAsJog`

The jog exemption for a Z-shaped jog also applies to an L-shaped jog.

Type: Boolean

`'useEdgeLength`

(Virtuoso Advanced Node for Layout Only) The length of the edges in the wrong-way direction (non-preferred routing direction) of a wrong-way jog in a Z shape must be less than *length*. Otherwise, the span (length of the jog) must be less than *length*.

Type: Boolean

`'horizontalJog | 'verticalJog`

The direction in which the length of the jog is measured. For example, if the routing direction is vertical, the jog direction must be horizontal.

Type: Boolean

`'jogWidth f_jogWidth f_wireWidth`

The constraint does not apply to a Z- or L-shaped jog with length less than *length* in the specified direction if the following conditions are satisfied:

- The width of the jog is less than or equal to *jogWidth*.
- The end-of-line width of the adjacent wires and the neighboring wire is less than or equal to *wireWidth*.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'eolWidth f_eolWidth`

The constraint value table does not apply to edges with width less than this value. Instead, the spacing specified with `'eolSpacingTable` applies.

`'endOfLineExemptionRanges g_endOfLineExemptionRanges`

The constraint value table applies to an end-of-line width that is less than `eolWidth` if it lies in the specified range.

Type: Floating-point values specifying a range of end-of-line widths to which the constraint applies

`'exactSpacingTable g_exactSpacingTable`

The spacing between two shapes can be equal to a spacing value listed in this table.

Type: A 1-D table, indexed on span, specifying floating-point spacing values

`'spacingToMinSpanTable g_spacingToMinSpanTable`

(Virtuoso Advanced Node for Layout Only) The spacing of a shape with a larger span to a shape with smaller span can be greater than or equal to a spacing value listed in this table if the larger of the two spans is greater than the corresponding index.

Type: A 1-D table, indexed on span, specifying floating-point spacing values

`'exactSelfSpacingTable g_exactSelfSpacingTable`

(Virtuoso Advanced Node for Layout Only) The spacing for shapes in the same span length range can be equal to a spacing value listed in this table.

Type: A 1-D table, indexed on span, specifying floating-point spacing values

`'eolSpacingTable g_eolSpacingTable`

The end-of-line spacing that is applied when the end-of-line width of the wire is less than `eolWidth`.

Type: A 1-D table, indexed on span, specifying floating-point spacing values

`'sameMask | 'diffMask`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Advanced Node for Layout Only) When `sameMask` is specified, the constraint applies only to shapes on the same mask, regardless of any different mask shapes that may be placed between them.

(Virtuoso Layout Suite EXL and higher tiers) When `diffMask` is specified, the constraint applies only between shapes of different mask colors.

Type: Boolean

`'minSpanSpacingRanges g_minSpanTable`

The additional exact spacings or a spacing range specified for a set of spans if the parallel run length between the shapes is non-zero.

Typically, these ranges are used to specify a different set of legal spacing ranges for wires with smaller spans. Therefore, this spacing is smaller than the spacing stored in the constraint's value table, which is based on wider span and parallel run length values.

The format of the table is as follows:

`(f_spanLength1 f_spanLength2) l_ranges`

where,

- `f_spanLength1` applies if the span of the first shape is greater than or equal to this value.
- `f_spanLength2` applies if the span of the second shape is greater than or equal to this value.
- `l_ranges` specifies the spacing required between the two shapes.

This parameter is used instead of

`'spacingToMinSpanTable` if spacing ranges need to be specified; it is mutually exclusive with both `'exactSpacingTable` and `'spacingToMinSpanTable`.

For information about how to interpret this parameter, see [Example 4a: minSpanLengthSpacing with minSpanSpacingRanges](#).

Type: A 2-D table, indexed on the span of the two shapes, specifying floating-point spacing values

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'minSpanSpacingRangesPr1 g_minSpanPr1Table`

(Virtuoso Advanced Node for Layout Only) The additional exact spacings or a spacing range specified by `'minSpanSpacingRanges` is applied if the parallel run length between the shapes is greater than the value specified by this parameter.

The format of the table is as follows:

$(f\_spanLength1\ f\_spanLength2)\ f\_pr1$

where,

- $f\_spanLength1$ applies if the span of the first shape is greater than or equal to this value.
- $f\_spanLength2$ applies if the span of the second shape is greater than or equal to this value.
- f_pr1 specifies the parallel run length required between the two shapes for the spacing values specified by `'minSpanSpacingRanges` to apply. The parallel run length between the two shapes must be greater than this value.

Type: A 2-D table, indexed on the span of the two shapes, specifying floating-point parallel run length values

`'minSpanForExceptEolWidth f_minSpanForExceptEolWidth`

(Virtuoso Advanced Node for Layout Only) The constraint applies to edges with width less than `eolWidth` if the span of a shape measured in the specified spacing direction is less than this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'eolExceptionNoPrl (Virtuoso Advanced Node for Layout Only) The exception of
eolWidth applies even when there is no PRL between two
shapes. If this parameter is not set, the eolWidth exception is
applied only when two shapes have common run length.
```

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Examples

- Example 1: minSpanLengthSpacing with horizontal, exceptJogLength, and exactSpacingTable
- Example 2: minSpanLengthSpacing with horizontal and exactSpacingTable
- Example 3: minSpanLengthSpacing with horizontal, exactSpacingTable, and spacingToMinSpanTable
- Example 4a: minSpanLengthSpacing with minSpanSpacingRanges
- Example 4b: minSpanLengthSpacing with minSpanSpacingRanges
- Example 4c: minSpanLengthSpacing with minSpanSpacingRanges
- Example 5: minSpanLengthSpacing with minSpanSpacingRanges and minSpanSpacingRangesPrI
- Example 6: minSpanLengthSpacing with eolWidth and eolSpacingTable

Virtuoso Technology Data Constraints Reference

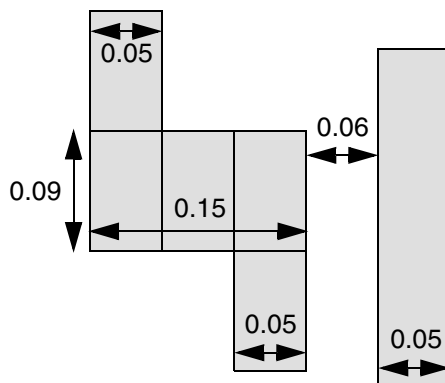
Spacing Constraints (One Layer)

Example 1: *minSpanLengthSpacing* with *horizontal*, *exceptJogLength*, and *exactSpacingTable*

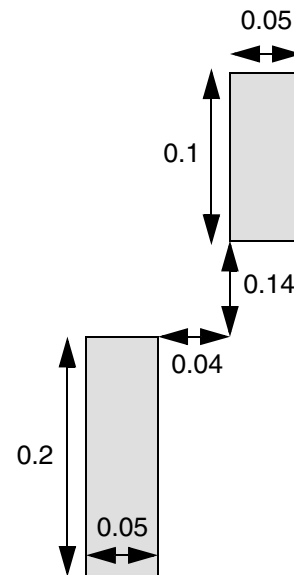
- The minimum horizontal spacing between two wires must be as follows:
 - If the larger of the two spans is less than 0.14, the spacing must be at least 0.06. An exact spacing of 0.05 is allowed.
 - If the larger of the two spans is greater than or equal to 0.14, the spacing must be at least 0.1.
- A Z-shaped jog is exempt from the spacing requirement if its length is less than 0.17 in the horizontal direction, width is less than or equal to 0.09, and the width of the adjacent wires and the neighboring wire perpendicular to the jog direction is less than or equal to 0.05.

```
spacingTables (
  ( minSpanLengthSpacing "Metal2"
    (( "pr1" nil nil "spanLength" nil nil )
      'horizontal
      'eolWidth 0.1
      'exceptJogLength 0.17
      'horizontalJog
      'jogWidth (0.09 0.05)
      'exactSpacingTable ((( "width" )) (0.00 0.05))
    )
    (
      (0.00 0.00) 0.06
      (0.00 0.14) 0.10
    )
  )
) ;spacingTables
```

□ Metal2



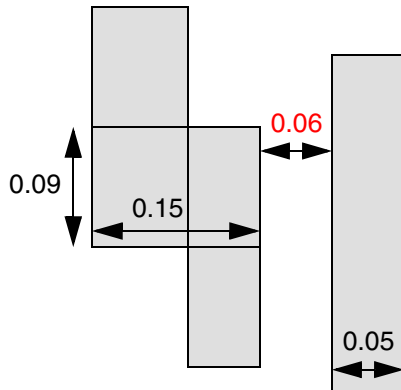
a) The constraint does not apply because the length of the Z-shaped jog in the horizontal direction is 0.15 (< 0.17), the width is 0.09 (≤ 0.09), and the width of the adjacent wires and the neighboring wire is 0.05 (≤ 0.05).



b) The constraint does not apply if the parallel run length is less than zero.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



c) FAIL. The projected parallel run length of the adjacent wires in the direction of the jog is zero, so the 'exceptJogLength' exception does not apply. The spacing between the two wires is 0.06, which is less than the required spacing of 0.1 for a span ≥ 0.14 .

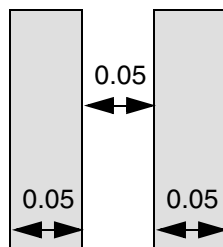
Example 2: minSpanLengthSpacing with horizontal and exactSpacingTable

The minimum horizontal spacing between two wires must be as follows:

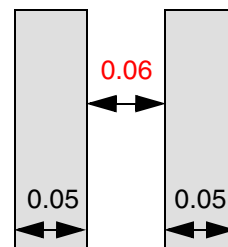
- If the larger of the two spans is less than 0.1, the spacing must be at least 0.07. An exact spacing of 0.05 is also acceptable.
- If the larger of the two spans is greater than or equal to 0.1, the spacing must be at least 0.11.

```
spacingTables (
  ( minSpanLengthSpacing "Metal2"
    (( "prl" nil nil "spanLength" nil nil )
     'horizontal
     'exactSpacingTable ((( "width" )) (0.00 0.05))
    )
    (
      (0.00 0.0) 0.07
      (0.00 0.1) 0.11
    )
  )
) ;spacingTables
```

□ Metal2



a) PASS. An exact spacing of 0.05 is allowed.



b) FAIL. The spacing between wires must be equal to 0.05 or greater than or equal to 0.07. A spacing of 0.06 is not allowed.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 3: minSpanLengthSpacing with horizontal, exactSpacingTable, and spacingToMinSpanTable

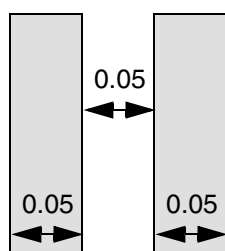
The minimum horizontal spacing requirement between two wires is as follows:

- For wires with span less than or equal to 0.09, the spacing must be at least 0.07 or equal to 0.05.
- For a wire with span greater than 0.09 and less than or equal to 0.11 to a wire with span less than or equal to 0.09, the spacing must be at least 0.06.
- For wires with span greater than 0.09 and less than or equal to 0.11, the spacing must be at least 0.08.
- For the larger of the two spans greater than 0.11, the spacing must be at least 0.09.

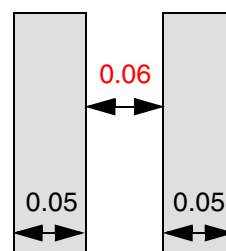
```
spacingTables ( □ Metal2
  ( minSpanLengthSpacing "Metal2"
    (( "pr1" nil nil "spanlength" nil nil )
      'horizontal
      'sameMask
      'exactSpacingTable ((( "spanlength" ))
        (
          0.00 0.05
          0.09 0.08
          0.11 0.09
        )
      )
    'spacingToMinSpanTable ((( "spanlength" ))
      (
        0.00 0.07
        0.09 0.06
        0.11 0.09
      )
    )
    (
      (0.00 0.00) 0.07
      (0.00 0.09) 0.08
      (0.00 0.11) 0.09
    )
  )
) ;spacingTables
```

Virtuoso Technology Data Constraints Reference

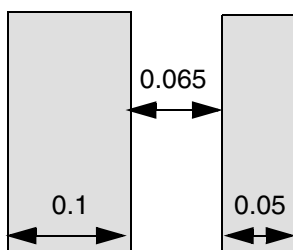
Spacing Constraints (One Layer)



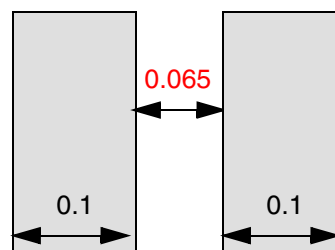
a) PASS. An exact spacing of 0.05 is allowed.



b) FAIL. The spacing between wires must be equal to 0.05 or at least 0.07. A spacing of 0.06 is not allowed.



c) PASS. The spacing between a span of 0.1 to a span of 0.05 must be at least 0.06.



d) FAIL. The spacing between the two wires must be at least 0.08.

Example 4a: minSpanLengthSpacing with minSpanSpacingRanges

The values used in the example are defined as follows:

- SpnN: The span of a shape measured in the spacing direction. Spacing is dependent on the span of one or both shapes.
$$\text{Spn1} < \text{Spn2} < \text{Spn3} < \dots < \text{SpnN}$$
- SN: The minimum spacing between two shapes measured in the spacing direction. This spacing is measured using Manhattan style spacing.
$$\text{S1} < \text{S2} < \text{S3} < \dots < \text{SN}$$
- X1, X2, ..., XN: The exact spacing required between two shapes.
- eolW: The edge length for which the constraint applies when the shape's span is greater than minSpanForExceptEolWidth.
- -prl1 < 0 < prl2: The parallel run length (prl) between shapes, ranging from a negative number (-prl1) to a positive number (prl2).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The following table represents the constraint's value table. In most cases, the spacing requirement for parallel run length 'p' is 0 if $-prl1 < p < 0$. However, this does not apply to the two smallest span lengths, where a minimum spacing requirement exists. Additionally, the rows for Sp2, Sp6, and Sp7 can be eliminated because the spacing values for these are the same as that in the neighboring rows (Sp1 and Sp5).

spanlength/ prl	-prl1 < 0	prl = 0	prl2 > 0
Sp1	S12	S12	S12
Sp2	S12	S12	S12
Sp3	0	S4	S4
Sp4	0	S7	S7
Sp5	0	S10	S10
Sp6	0	S10	S10
Sp7	0	S10	S10
Sp8	0	S13	S10
Sp9	0	S11	S14

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

The table below represents the spacing table for the 'minSpanSpacingRanges parameter. The table stops at shapes with span Sp6. This is because the spacing for shapes with span Sp7 and up depends only on the parallel run length, which means that the constraint's value table holds all the required information.

spanLength/ spanLength	Sp1	Sp2	Sp3	Sp4	Sp5	Sp6
Sp1	=X1, =X2, = X3, >=S12	=X1, =X2, =X3, >=S12	>=S9	>=S3	>=S8	>=S7
Sp2	=X1, =X2, =X3, >=S12	=X1, =X2, =X3, >=S12	>=S9	>=S3	>=S8	>=S7
Sp3	>=S9	>=S9	0	0	0	0
Sp4	>=S3	>=S3	0	0	0	0
Sp5	>=S8	>=S8	0	0	0	0
Sp6	>=S7	>=S7	0	0	0	0

To reduce duplication of data, entries in this table that duplicate information in the constraint's value table can be set to 0. For example, the spacing between shapes of Sp3 and Sp4 is >=S7, which is the same as that in the Sp4 row of the constraint's value table for prl > 0. Therefore, the corresponding entry in this table is set to 0.

Additionally, row Sp1 in the table above can be eliminated because rows Sp1 and Sp2 are identical. Similarly, column Sp1 can be eliminated because columns Sp1 and Sp2 are identical.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

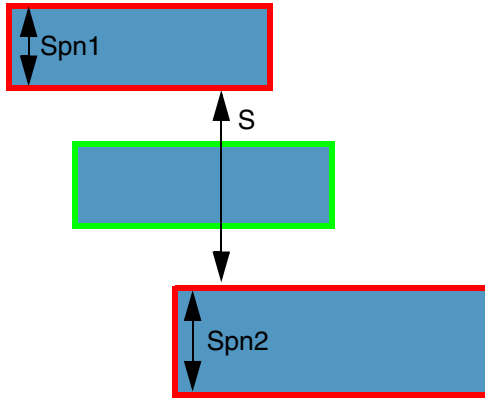
```
spacingTables(
  ( minSpanLengthSpacing "Metal1"
    ("pr1" nil nil "spanLength" nil nil)
    'vertical
    'sameMask
    'eolWidth eolW
    'minSpanForExceptEolWidth mSFEEW
    'minSpanSpacingRanges ((( "spanLength1" nil nil
                               "spanLength2" nil nil ) 0.0)
      (
        (sp1 sp1) (X1 X2 X3 ">= S12")
        (sp1 sp2) (X1 X2 X3 ">= S12")
        (sp2 sp1) (X1 X2 X3 ">= S12")
        (sp2 sp2) (X1 X2 X3 ">= S12")
        (sp1 sp3) ( ">= S9")
        (sp2 sp3) ( ">= S9")
        (sp3 sp1) ( ">= S9")
        (sp3 sp2) ( ">= S9")
        (sp1 sp4) ( ">= S3")
        (sp2 sp4) ( ">= S3")
        (sp4 sp1) ( ">= S3")
        (sp4 sp2) ( ">= S3")
        (sp1 sp5) ( ">= S8")
        (sp2 sp5) ( ">= S8")
        (sp5 sp1) ( ">= S8")
        (sp5 sp2) ( ">= S8")
        (sp1 sp6) ( ">= S7")
        (sp2 sp6) ( ">= S7")
        (sp6 sp1) ( ">= S7")
        (sp6 sp2) ( ">= S7")
      )
    )
  )
  (
    (-pr11 sp1) S12
    (0 sp1) S12
    (pr12 sp1) S12
    (-pr11 sp2) S12
    (0 sp2) S12
    (pr12 sp2) S12
    (0 sp3) S4
    (pr12 sp3) S4
    (0 sp4) S7
    (pr12 sp4) S7
    (0 sp5) S10
    (pr12 sp5) S10
    (0 sp6) S10
    (pr12 sp6) S10
    (0 sp7) S10
    (pr12 sp7) S10
    (0 sp8) S13
    (pr12 sp8) S10
    (0 sp9) S11
    (pr12 sp9) S14
  )
) ;spacingTables
```

 Metal1
 mask1
 mask2

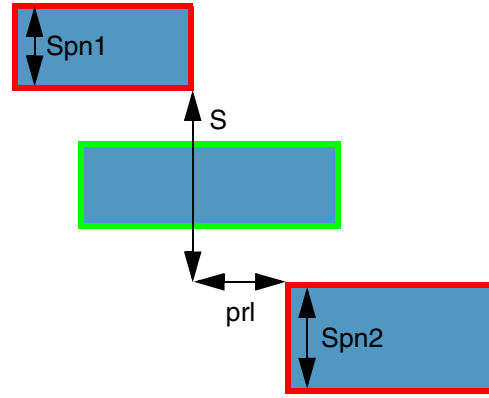
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

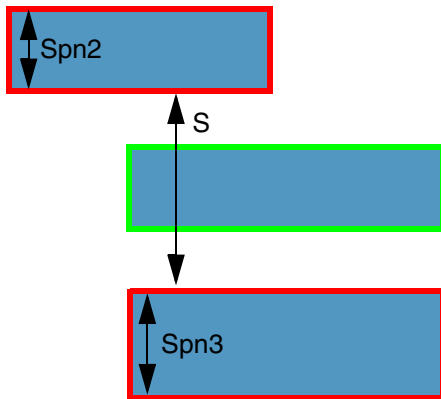
Figures (a), (b), (c), and (d) illustrate how the '`minSpanSpacingRanges`' parameter is used to specify a set of spacing ranges for wires with smaller spans.



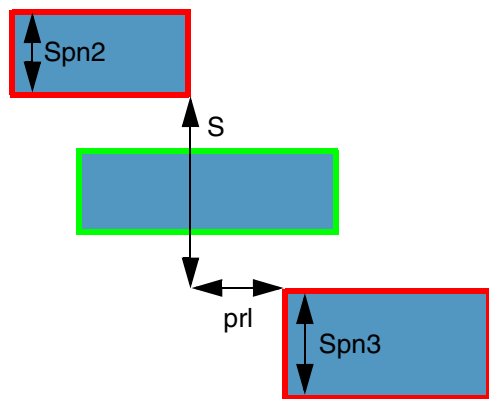
a) $prl > 0$; fails if $S \neq X1, X2$, or $X3$ and is $< S12$; passes if $S \geq S12$.



b) $prl < 0$; the constraint does not apply if $prl < -prl1$; fails if $prl \geq -prl1$ and $S \neq X1, X2$, or $X3$ and is $< S12$.



c) $prl > 0$; fails if $S < S4$; passes if $S \geq S4$

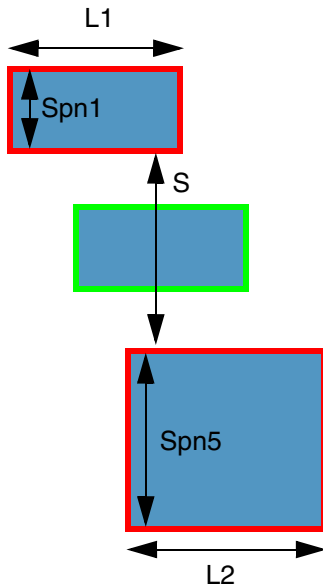


d) $prl < 0$; passes even if $S < S4$.

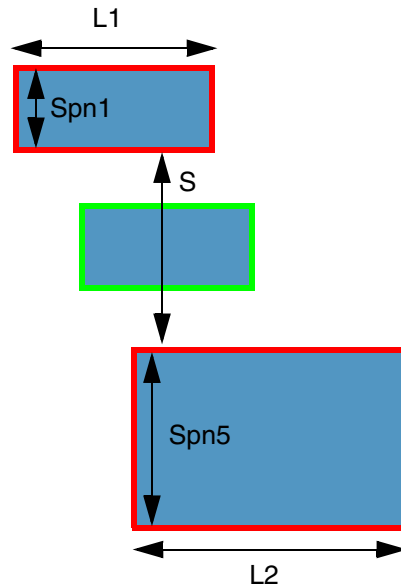
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

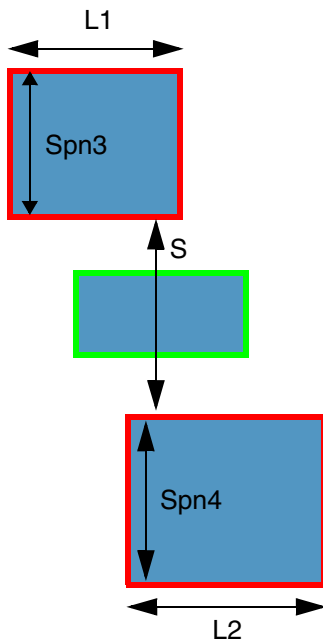
Figures (e), (f), and (g) illustrate how spacing is calculated for wider span wires by using both 'eolWidth and 'minSpanForExceptEolWidth parameters.



e) $prl > 0$, $L1 < eolW$, $L2 < eolW$
The constraint does not apply because neither $L1$ nor $L2$ is greater than $eolW$.



f) $prl > 0$, $L1 < eolW$, $L2 > eolW$
The constraint applies because $L2 > eolW$. The constraint's value table requires $S > S10$. However, the span of the top shape is $Spn1$ and there is a nonzero entry of $\geq S8$ in the `minSpanSpacingRanges` table for $Spn5/Spn1$. Therefore, if $S \geq S8$, the constraint is met.



g) $prl > 0$, $L1 < eolW$, $L2 < eolW$, $Spn4 < minSpanForExceptEolWidth$
The constraint applies even though $L1$ and $L2$ are both less than $eolW$ because $Spn4$ is less than `minSpanForExceptEolWidth`. If $S \geq S7$, the constraint is met.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

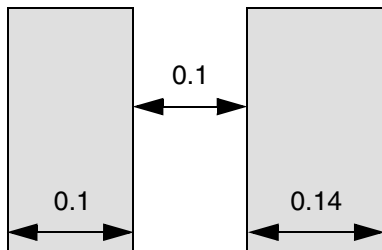
Example 4b: minSpanLengthSpacing with minSpanSpacingRanges

The minimum spacing requirement between two wires is as follows:

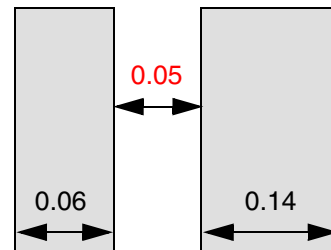
- If the larger of the two spans is less than 0.14, the spacing must be greater than or equal to 0.05.
- If the larger of the two spans is greater than or equal to 0.14, the spacing must be greater than or equal to 0.1.

```
spacingTables(
  ( minSpanLengthSpacing "Metal2"
    (( "pr1" nil nil "spanLength" nil nil )
      'minSpanSpacingRanges
      (
        (( "spanLength1" "spanLength2" ))
        (
          (0.060 0.068) (0.05)
          (0.100 0.100) (0.05)
        )
      )
    )
  )
  (
    (0.00 0.00) 0.05
    (0.00 0.14) 0.10
  )
)
) ;spacingTables
```

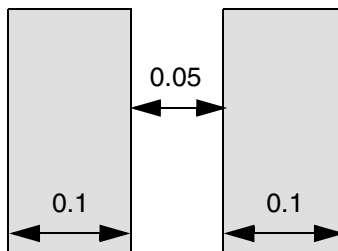
□ Metal2



a) PASS. The spacing between a span length of 0.1 to a span length of 0.14 must be at least 0.1.



b) FAIL. The spacing between a span length of 0.06 to a span length of 0.14 must be at least 0.1.



c) PASS. The spacing between two wires with span 0.1 must be at least 0.05.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 4c: minSpanLengthSpacing with minSpanSpacingRanges

The minimum spacing requirement between two wires when parallel run length is greater than or equal to -0.1 is as follows:

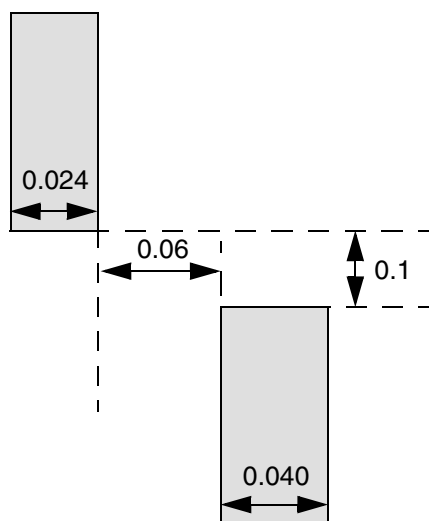
- If one span is greater than or equal to 0.0235 and the other span is greater than or equal to 0.0395, the spacing between the wires must be equal to 0.06 or greater than 0.1.
- If the larger of the two spans is less than 0.0395, the spacing must be greater than or equal to 0.12.
- If the larger of the two spans is greater than or equal to 0.0395, the spacing must be greater than or equal to 0.1.

```
spacingTables(  
  ( minSpanLengthSpacing "Metal2"  
    (( "pr1" nil nil "spanLength" nil nil )  
      'horizontal  
      'minSpanSpacingRanges  
        (  
          (( "spanLength1" "spanLength2" ))  
          (  
            (0.0235 0.0395) (0.06 "> 0.1")  
            (0.0395 0.0235) (0.06 "> 0.1")  
          )  
        )  
      )  
    )  
    (  
      (-0.1 0.0235) 0.12  
      (-0.1 0.0395) 0.10  
    )  
  )  
) ;spacingTables
```

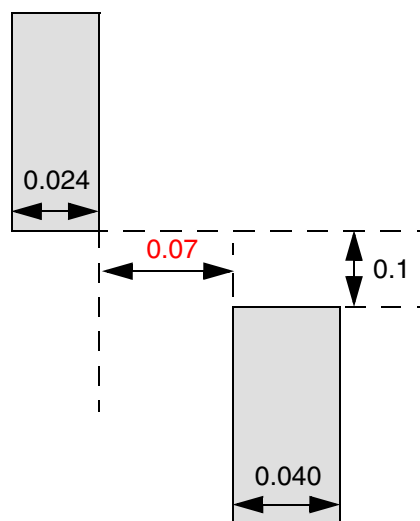
☐ Metal2

Virtuoso Technology Data Constraints Reference

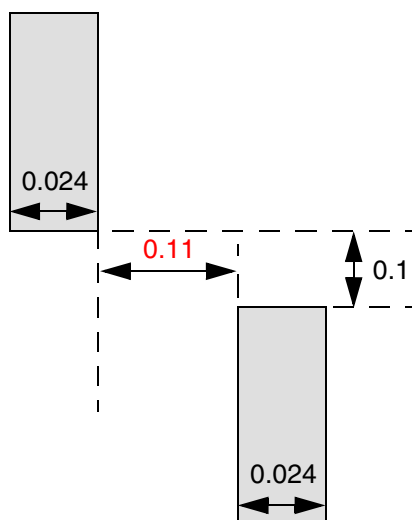
Spacing Constraints (One Layer)



a) PASS. An exact spacing of 0.06 is allowed.



b) FAIL. The spacing between the wires must be equal to 0.06 or greater than 0.1. A spacing of 0.07 is not allowed.



c) FAIL. The spacing between the two wires with span length 0.024 must be at least 0.12.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 5: minSpanLengthSpacing with minSpanSpacingRanges and minSpanSpacingRangesPr1

The minimum spacing requirement between two wires is as follows:

- If both spans are greater than or equal to 0 and the parallel run length is equal to -0.01, the spacing between the wires must be equal to 0.05 or greater than or equal to 0.07.
- If the larger of the two spans is greater than or equal to 0.09 and the parallel run length is equal to -0.02, the spacing must be equal to 0.06 or greater than or equal to 0.08.
- If the larger of the two spans is greater than or equal to 0.09 and the parallel run length is equal to -0.03, the spacing must be equal to 0.08 or greater than or equal to 0.08.
- If both spans are greater than or equal to 0.09 and the parallel run length is equal to 0, the spacing between the wires must be greater than or equal to 0.08.

```
spacingTables (
  ( minSpanLengthSpacing "Metal2"
    (( "pr1" nil nil "spanlength" nil nil )
      'vertical
      'exceptJogLength 0.1
      'useEdgeLength
      'verticalJog
      'jogWidth (0.09 0.05)
      'minSpanSpacingRanges
      (
        (( "spanLength1" nil nil "spanLength2" nil nil ))
        (
          (0.0 0.0 ) (0.05 ">=0.07")
          (0.0 0.09) (0.06 ">=0.08")
          (0.09 0.0 ) (0.08 ">=0.08")
          (0.09 0.09) (">=0.08")
        )
      )
    )
    'minSpanSpacingRangesPr1
    (
      (( "spanLength1" nil nil "spanLength2" nil nil ))
      (
        (0.0 0.0 ) -0.01
        (0.0 0.09) -0.02
        (0.09 0.0 ) -0.02
        (0.09 0.09) 0
      )
    )
  )
  (
    (0.0 0.0 ) 0.07
    (0.0 0.09) 0.08
  )
) ;spacingTables
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 6: minSpanLengthSpacing with eolWidth and eolSpacingTable

If the end-of-line width of a wire is less than 0.05, the spacing requirement between this end-of-line wire and another wire, as determined by their span, is as follows:

- If both have spans greater than or equal to 0, the end-of-line spacing must be at least 0.07.
- If both have spans greater than or equal to 0.09, the end-of-line spacing must be at least 0.07.
- If both have spans greater than or equal to 0.11, the end-of-line spacing must be at least 0.09.

```
spacingTables(  
  ( minSpanLengthSpacing "Metal2"  
    (( "pr1" nil nil "spanlength" nil nil )  
      'vertical  
      'eolWidth 0.05  
      'eolSpacingTable ((( "spanlength" ))  
        (  
          0.0    0.07  
          0.09   0.07  
          0.11   0.09  
        )  
      )  
    )  
  )  
  (  
    (0.0  0.0 )  0.07  
    (0.0  0.09)  0.08  
    (0.0  0.11)  0.09  
  )  
) ;spacingTables
```

minStubInfluenceSpacing

```

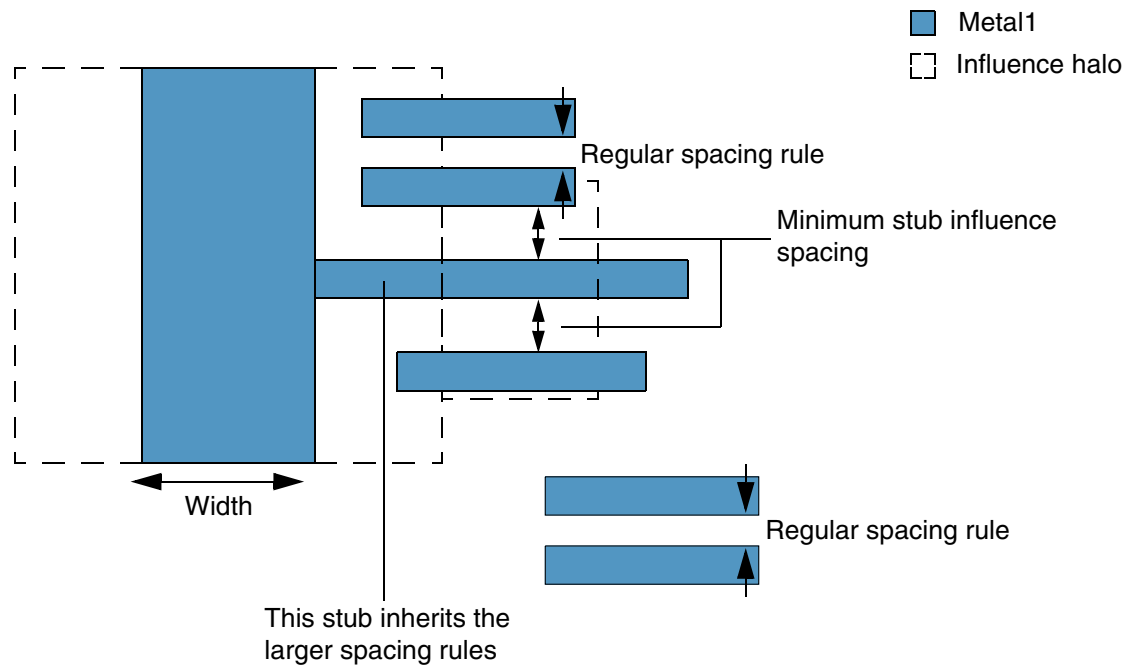
spacings (
    ( minStubInfluenceSpacing tx_layer
      'width f_stubWidth
      f_spacing
    )
) ;spacings

spacingTables (
    ( minStubInfluenceSpacing tx_layer
      (( "width" nil nil ["distance" nil nil] )
      'width f_stubWidth
      [f_default]
    )
    (g_table)
  )
) ;spacingTables

```

Specifies the minimum spacing between a stub shape (a protrusion from a wide wire) and a neighboring shape that is on the same layer, but on a different net, when it is within a certain distance of the large shape from which the stub protrudes.

Also known as the proximity or influence rule, the spacing required by the stub shape is influenced by the large shape to which it is connected.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the stub shape and the shapes inside the influence halo must be greater than or equal to this value.
"width" nil nil ["distance" nil nil]	This identifies the index for <i>table</i>.
<i>g_table</i>	The format of a row in a 1-D <i>table</i> is as follows: <i>(f_width f_spacing)</i> where, <i>f_width</i> is the width of the large influencing shape and <i>f_spacing</i> is the minimum spacing required between the stub shape and the shapes inside the influence halo when the width of the large influencing shape is greater than or equal to the corresponding index. The influence halo extends to the full length of the stub shape. The format of a row in a 2-D <i>table</i> is as follows: <i>(f_width f_distance) f_spacing</i> where, ■ <i>f_width</i> is the width of the large influencing shape and <i>f_distance</i> is the distance up to which the influence halo extends from the large influencing shape. ■ <i>f_spacing</i> is the minimum spacing required between the stub shape and the shapes inside the influence halo when both width and distance are greater than or equal to the corresponding indexes. Type: A 1-D table specifying floating-point width and spacing values, or a 2-D table specifying floating-point width, distance, and spacing values

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'width f_stubWidth`

The constraint applies if the width of the short stub wire is less than or equal to this value.

`f_default`

The spacing value to be used when no table entry applies.

Example

The minimum influence spacing on Poly1 must be at least 0.5, and the minimum spacing on Metal2 must be at least equal to the value of the technology parameter `minspacing1`.

```
spacings (
  ( minStubInfluenceSpacing "Poly1"
    0.5
  )
  ( minStubInfluenceSpacing "Metal2"
    techParam("minspacing1")
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

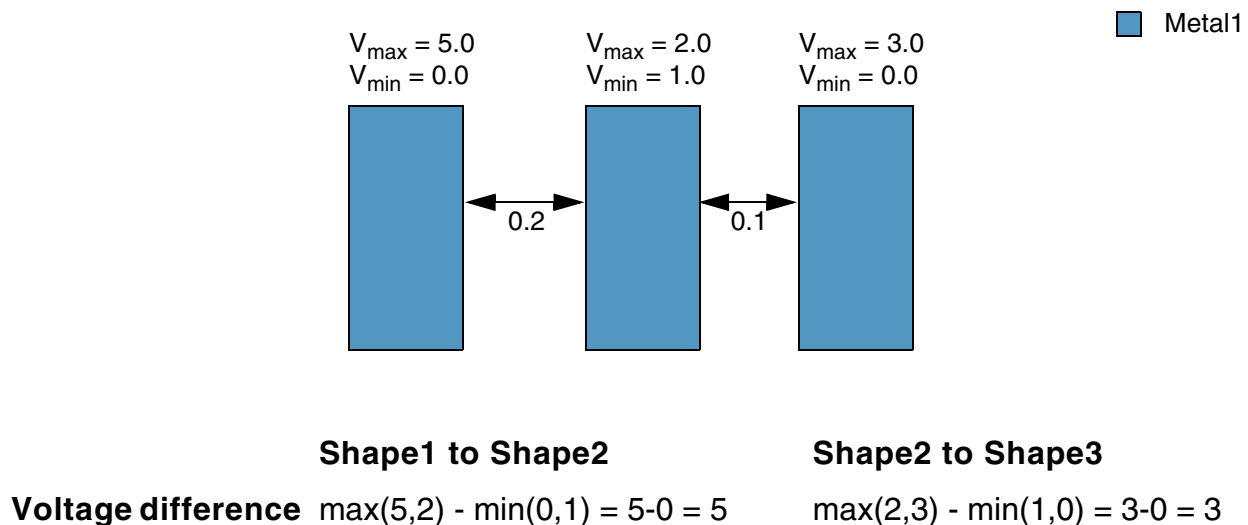
minVoltageSpacing (One layer)

```
spacingTables(  
  ( minVoltageSpacing tx_layer  
    (( "voltage" nil nil ))  
    ['horizontal | 'vertical]  
    (g_table)  
  )  
) ;spacingTables
```

Specifies the minimum spacing between two wires on a layer when the maximum difference in voltage between the two wires is greater than or equal to the specified voltage. The difference in voltage is calculated as follows:

$$V_{\text{diff}} = \max(V_{\text{max}} \text{ of each shape}) - \min(V_{\text{min}} \text{ of each shape})$$

Note: Layer-purpose pairs are not supported for this rule. If a layer-purpose pair is found, a warning is displayed and the constraint is created as a non-purpose-aware constraint, that is, the purpose is reassigned as `techcNoPurpose`.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<code>"voltage" nil nil</code>	This identifies the index for <i>table</i> .
<code>g_table</code>	The format of the <i>table</i> row is as follows: $(f\_voltage\ f\_distance)$ where, ■ <i>f_voltage</i> is the voltage difference between two wires. The voltage difference must be greater than or equal to this value. ■ <i>f_distance</i> is the minimum required spacing. Type: A 1-D table specifying floating-point voltage and spacing values

Parameters

<code>'horizontal 'vertical</code>	(Virtuoso Advanced Node for Layout Only) The direction in which spacing is measured. If the direction is not specified, spacing is measured in any direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

The spacing between two Metal1 wires must be at least:

- 0.065 if the maximum voltage difference between the wires is greater than or equal to 0.0V and less than 1.5V
- 0.085 if the maximum voltage difference between the wires is greater than or equal to 1.5V and less than 3.3V
- 0.105 if the maximum voltage difference between the wires is greater than or equal to 3.3V

It must be measured in the vertical direction.

```
spacingTables(  
  ( minVoltageSpacing "Metal1"  
    (( "voltage" nil nil ))  
    (  
      0.0 0.065  
      1.5 0.085  
      3.3 0.105  
    )  
    'vertical  
  )  
) ;spacingTables
```

oneSideSpacing

```
spacings (
    ( oneSideSpacing (tx_layer1 tx_layer2 ... tx_layerN) )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies that the `minSpacing` rules defined in the default constraint group of a net apply to any one side of the shapes on that net.

Values

tx_layer1 tx_layer2 ... tx_layerN

The layers on which the constraint is applied. Use a space-separated list of layer names or layer numbers. Additionally, enclose each layer name in double quotes.

Type: String (layer name) or Integer (layer number)

Parameters

None

Examples

- Example 1: oneSideSpacing with layer names
- Example 2: oneSideSpacing with layer numbers

Example 1: oneSideSpacing with layer names

The `minSpacing` rules defined in the default constraint group of a net apply to any one side of Metal1, Metal2, and Metal3 shapes on that net.

```
spacings (
    ( oneSideSpacing ("Metal1" "Metal2" "Metal3") )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 2: oneSideSpacing with layer numbers

The `minSpacing` rules defined in the default constraint group of a net apply to any one side of Metal1, Metal2, and Metal3 shapes on that net.

```
spacings (
  ( oneSideSpacing (30 34 38) )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

onGridOnly

```
spacings (
    ( onGridOnly tx_layer
      ['checkMask]
      ['exceptWidth g_exceptWidth]
      ['exceptVia]
      {"any" | "horizontal" | "vertical"}

    )
); spacings
```

(Virtuoso Advanced Node for Layout Standard) Requires that wires in the specified direction, on the specified layer, must be on the grid or tracks.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"any" "horizontal" "vertical"	The direction of the wires.

Parameters

'checkMask	The wire color should be checked against the track color. For wires with the default width, the wire and track color should be the same. If wide wires, by taking the reverse track color, can block fewer tracks based on the spacing, the wire color should be the reverse of the track color. Type: Boolean
'exceptWidth <i>g_exceptWidth</i>	The on-grid wire checking does not apply if the width of the wire is more than the given value.
'exceptVia	(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the metal shape of a via is not checked against the rule.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

Wires in the vertical direction must be on the grid.

```
spacings (
  ( onGridOnly "Metal1"
    'checkMask
    'exceptWidth 0.05
    "vertical"
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

pgViaTrack

```
spacings(  
    ( pgViaTrack tx_cutLayer  
      'coreOffset f_offset  
      {'horizontalTrack | 'verticalTrack}  
      f_pitch  
    )  
); spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the pitch of the tracks on *cutLayer*, in the specified direction.

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

'coreOffset <i>f_offset</i>	The starting coordinate of the track with respect to the origin of the core.
'horizontalTrack 'verticalTrack	The direction of the tracks. Type: Boolean

Virtuoso Technology Data Constraints Reference

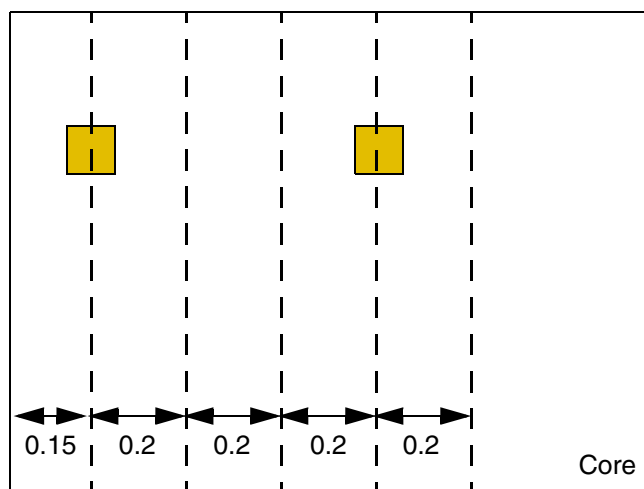
Spacing Constraints (One Layer)

Example

The tracks on Via1 are defined with the starting coordinate of 0.01 with respect to the origin of the core and a repeatable pitch of 0.2.

```
spacings(  
  ( pgViaTrack "Via1"  
    'coreOffset 0.15  
    'verticalTrack  
      0.2  
  )  
) ;spacings
```

■ Via1



The via cuts on Via1 must align to the specified tracks.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

requiredEndOfLineShape

```
spacings (
    ( requiredEndOfLineShape tx_layer
      'shapeWidthRange f_shapeWidthRange
      'lengthRange f_lengthRange
      ['horizontal | 'vertical]
      f_spacingRange
    )
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the spacing at which at least one end-to-end shape must be present on at least one end of a rectangular shape.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacingRange</i>	The spacing at which an end-to-end shape must be present on at least one of the ends of a rectangular shape.

Parameters

<i>'shapeWidthRange f_shapeWidthRange</i>	The width of the rectangular shape on which the constraint applies must be within this range. Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint
<i>'lengthRange f_lengthRange</i>	The length of the rectangular shape on which the constraint applies must be within this range.
<i>'horizontal 'vertical</i>	The direction in which spacing between shapes is measured. If direction is not specified, spacing is measured in any direction. Type: Boolean

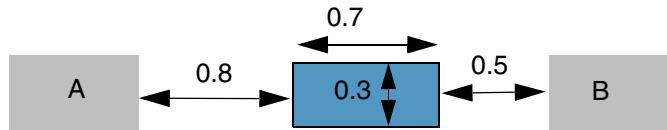
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

A rectangular shape whose width is less than or equal to 0.4 and length less than or equal to 0.8 must have at least one neighboring shape at a spacing of 0.5 measured in the horizontal direction.

```
spacings (                                     ■ Metal1
  ( requiredEndOfLineShape "Metal1"
    'shapeWidthRange "<=0.4"
    'lengthRange "<=0.8"
    'horizontal
    "=0.5"
  )
) ;spacings
```



PASS. The Metal1 shape has width 0.3 and length 0.7. To the right end is a shape marked B at an end-to-end spacing of 0.5.

If shape B was not at this spacing, the constraint would fail because shape A at the left end is at a spacing of 0.8, which is greater than 0.5.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

shapeRequiredBetweenSpacing

```
orderedSpacings (
  ( shapeRequiredBetweenSpacing tx_layer1 tx_layer2
    ['horizontal | 'vertical]
    f_spacing
  )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies that shapes on *layer1* that are at a distance less than the specified value must have a *layer2* shape between them, separating the entire length of the Metal1 shapes.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The constraint applies only if the distance between two <i>layer1</i> shapes is less than this value.

Parameters

<code>'horizontal 'vertical</code>	The direction in which spacing is measured between two <i>layer1</i> shapes. If direction is not specified, spacing is measured in any direction.

Virtuoso Technology Data Constraints Reference

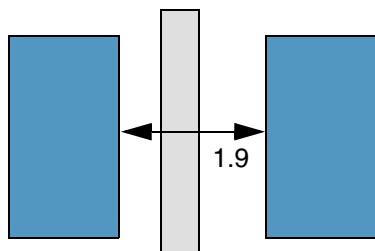
Spacing Constraints (One Layer)

Example

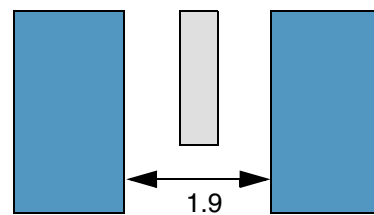
Two Metal1 shapes that are less than 2.0 apart in the horizontal direction must have a Metal2 shape between them, and this Metal2 shape must fully separate the two Metal1 shapes.

```
orderedSpacings (  
  ( shapeRequiredBetweenSpacing "Metal1" "Metal2"  
    'horizontal  
    2.0  
  )  
) ;orderedSpacings
```

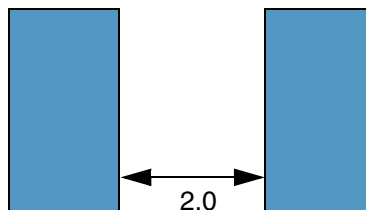
□ Metal2
■ Metal1



a) PASS. The horizontal distance between Metal1 shapes is 1.9 (< 2.0) and a Metal2 shape is present between the Metal1 shapes.



b) FAIL. The Metal2 shape does not fully separate the Metal1 shapes.



c) PASS. Metal1 shapes do not require a Metal2 shape between them because the horizontal distance between Metal1 shapes is equal to 2.0.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

sixWiresSpacing

```
spacings(  
  ( sixWiresSpacing tx_Layer  
    'eolSpacing f_eolSpacing  
    {'sameSide | 'oppositeSide}  
    {'horizontal | 'vertical}  
    'width f_width 'sideSpacing f_sideSpacing  
    g_spacing  
  )  
); spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies a spacing rule on a six-wire pattern.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_spacing</i>	The spacing range value. Type: Floating-point values specifying a <u>range</u> of spacings that are allowed

Parameters

<i>'eolSpacing g_eolSpacing</i>	The two perfectly aligned wires are less than or equal to the specified value on both of the two sets of middle wires.
<i>'sameSide 'oppositeSide</i>	Specifies the side of the two line-ends of the two wires .
<i>'horizontal 'vertical</i>	The direction in which the spacing is measured.
<i>'width f_width</i>	The width of all of the six wires should be equal to the specified value.
<i>sideSpacing f_sideSpacing</i>	The side-to-side spacing between any two wires in the perpendicular direction in <i>oaSpacingDirection</i> should be equal to the specified value.

Example

```
spacings(  
  (cornerEndOfLineKeepout "Metall"  
    'minWidth 0.05  
    'eolSpacing 0.1  
    'eolWithin (0.08 0.12)  
    (0.06 0.09))  
) ;spacings
```

snapPatternDefOffset

```
spacings (
    ( snapPatternDefOffset
      {'horizontal | 'vertical | 'snapPatternDef t_name}
      f_offset
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the snapPatternDef offset. The offset can be specified either for all snapPatternDefs in a particular direction or for a particular snapPatternDef.

The constraint applies to both snapPatternDefs and widthSpacingSnapPatternDefs.

Note: For widthSpacingSnapPatternDefs, this constraint overrides the offset of the widthSpacingSnapPatternDef, not the offset of the widthSpacingPattern. In other words, it shifts the period grid, not the tracks within the period.

Values

f_offset The offset value.

Parameters

'horizontal | 'vertical
The direction in which the offset is measured.

'snapPatternDef t_name
The snapPatternDef name to which the offset requirement applies.

Example

The offset required for M2_globalGrid snapPatternDef is 0.1.

```
spacings (
    ( snapPatternDefOffset
      'snapPatternDef "M2_globalGrid"
      0.1
    )
) ;spacings
```

threeShapesForbiddenSpacingRanges

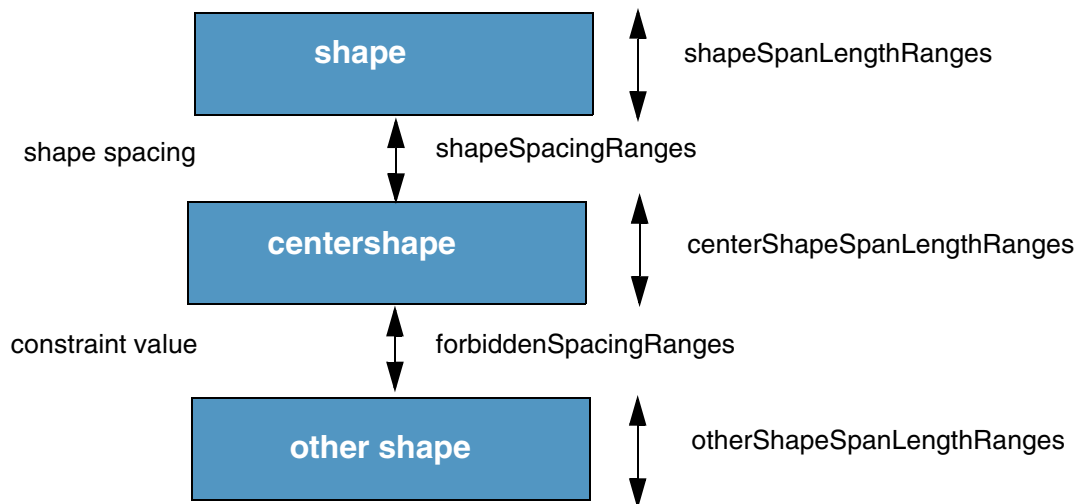
```

spacings (
  ( threeShapesForbiddenSpacingRanges tx_layer
    'horizontal | 'vertical
    'shapeSpacingRanges g_spacingRanges
    'centerShapeSpanLengthRanges g_centerShapeSpanLengthRanges
    ['shapeSpanLengthRanges g_shapeSpanLengthRanges]
    ['otherShapeSpanLengthRanges g_otherShapeSpanLengthRanges]
    (g_forbiddenOtherSpacingRanges)
  )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Specifies a list of forbidden spacing ranges between three shapes on a given layer.

■ Metal1



Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_forbiddenOtherSpacingRanges</i>	The spacing that is forbidden between the three shapes. Type: A list of floating-point value <u>ranges</u> specifying forbidden spacing.

Parameters

<code>'horizontal 'vertical</code>	The direction in which forbidden spacing is measured. Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

'shapeSpacingRanges *g_spacingRanges*

The constraint applies to these allowed spacing ranges from the shape at the center to the shape at one side.

'centerShapeSpanLengthRanges *g_centerShapeSpanLengthRanges*

The constraint applies to these span length ranges for the shape at the center.

'shapeSpanLengthRanges *g_shapeSpanLengthRanges*

The constraint applies to these span length ranges for a shape on one side of the center shape. The default span length is greater than zero and the rule applies to all shapes.

'otherShapeSpanLengthRanges *g_otherShapeSpanLengthRanges*

The constraint applies to these span length ranges for a shape on the other side of the center shape. The default value is greater than zero and the rule applies to all shapes.

Examples

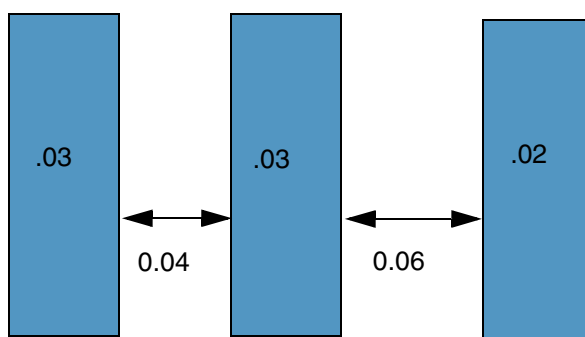
The spacing direction is specified as horizontal. The span length and spacing is measured in the spacing direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

```
(threeShapesForbiddenSpacingRanges "Metal1"  
  'horizontal  
  'shapeSpacingRanges      ("(0.03 0.05)")  
  'centerShapeSpanLengthRanges ("0.03")  
  'shapeSpanLengthRanges    ("0.03")  
  'otherShapeSpanLengthRanges ("0.02")  
  ("<=0.05" "(0.07 0.09)")  
)
```

■ Metal1



PASS: The constraint applies and both the shape spacing and the forbidden spacing are met.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

trimMinAdjacentSpacing

```
spacings (
    ( trimMinAdjacentSpacing tx_layer
      'count x_count
      'distance f_distance
      ['mask1 | 'mask2 | 'mask3 | 'mask4]
      f_spacing
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Defines the minimum spacing between adjacent trim shapes on the specified layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between adjacent trim shapes must be greater than or equal to this value.

Parameters

<i>'count x_count</i>	The constraint applies only if the number of adjacent trim shapes is greater than or equal to this value.
<i>'distance f_distance</i>	The distance between two trim shapes must be less than this value for them to be considered adjacent.
<i>'mask1 'mask2 'mask3 'mask4</i>	The mask of the trim layer to which the constraint is applied. This is used only for multi-patterned trim layers. Type: Boolean

Virtuoso Technology Data Constraints Reference

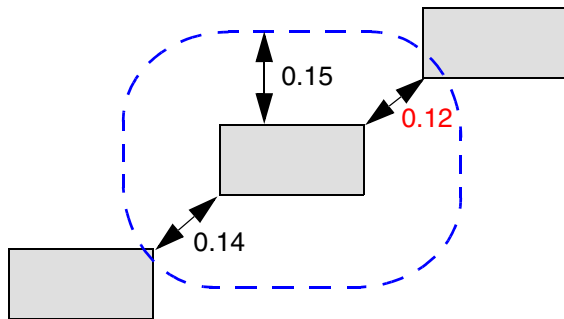
Spacing Constraints (One Layer)

Example

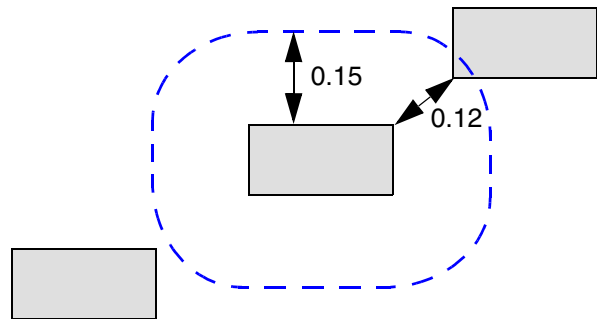
The distance between adjacent trim shapes must be at least 0.13 if a trim shape has at least two trim shapes at a distance less than 0.15 from it.

```
spacings(  
  ( trimMinAdjacentSpacing "TrimMetal1"  
    'count 2  
    'distance 0.15  
    0.13  
  )  
) ;spacings
```

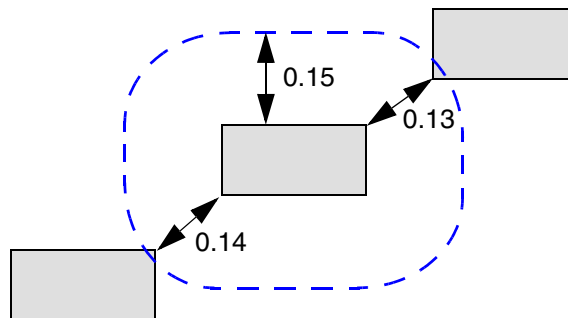
□ TrimMetal1



a) FAIL. There are two shapes at a distance less than 0.15 from the middle shape, but the distance between the middle and top shapes is only 0.12 (<0.13).



b) The constraint does not apply because there is only one shape at a distance less than 0.15 from the middle shape.



c) PASS. There are two shapes at a distance less than 0.15 from the middle shape and both are at a distance greater than or equal to 0.13 from the middle shape.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

trimMinSpacing (One layer)

```
spacings (
  ( trimMinSpacing tx_layer
    ['mergedTrimLength f_length]
    ['prlSpacing (f_spacing1 f_spacing2)]
    ['endToEndSpacing f_end['mergedTrimLength f_length]ToEndSpacing
      {'horizontal | 'vertical}
      ['prl f_prl]
      ['exactAligned f_exactAlignedSpacing]
      ['middleWireOnly f_middleWireLayer]
      ['edgeAligned f_edgeAlignedSpacing
        'exceptDiffMaskAligned]
      ['exceptWidth f_exceptWidth 'layer tx_exceptLayer]
    ]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['sameMask]
    ['mask1 | 'mask2 | 'mask3]
    f_spacing
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Defines the minimum spacing between shapes on the specified trim metal layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between shapes must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'mergedTrimLength f_length`

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the trim spacing applies only to a merged trim with length greater than or equal to the given value against another merged or non-merged trim.

`'prlSpacing (f_spacing1 f_spacing2)`

If the parallel run length between two shapes is zero, the spacing between them must be greater than or equal to *spacing1*; but if the parallel run length is greater than zero, the spacing between them must be greater than or equal to *spacing2*.

Otherwise, the spacing between the two shapes must be greater than or equal to *spacing*.

`'endToEndSpacing f_endToEndSpacing`

The 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 *prl*, when specified, or equal to zero.

`'horizontal | 'vertical`

The direction in which end-to-end spacing is measured.

Type: Boolean

`'prl f_prl`

The end-to-end spacing is applied only if the parallel run length between the shapes is greater than this value, when specified.

`'exactAligned f_exactAlignedSpacing 'exceptDiffMaskAligned`

The end-to-end spacing must be greater than or equal to this value if the shapes are aligned.

If *prl* is not specified, end-to-end spacing is applied in Euclidian measurement. If *prl* is specified, end-to-end spacing is applied in Manhattan measurement.

If *'sameMask* is also specified, *'exceptDiffMaskAligned* can be used to indicate that the constraint does not apply to two aligned trim shapes on the same mask when there is another exactly aligned trim shape on a different mask between them.

`'middleWireOnly f_middleWireLayer`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the trim spacing applies only if there is a metal shape on the specified layer overlapping the area between the trims.

`'edgeAligned f_edgeAlignedSpacing`

The end-to-end spacing must be greater than or equal to this value if the top or left edge of one shape is exactly aligned with the bottom or right edge of another shape.

Otherwise, *endToEndSpacing* is applied.

`'exceptWidth f_exceptWidth`

If a shape on *exceptLayer* is between two trim shapes and has width greater than or equal to this value, all end-to-end spacing rules are exempted.

`'layer tx_exceptLayer`

The layer on which `'exceptWidth` applies. This is the layer that the trim metal trims.

Type: String (layer name) or Integer (layer number)

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'sameMask`

The constraint applies only between shapes on the same mask.

Type: Boolean

`'mask1 | 'mask2 | 'mask3`

The constraint applies only to shapes on the specified mask.

Type: Boolean

Examples

- Example 1: trimMinSpacing with prlSpacing, endToEndSpacing, and vertical
- Example 2: trimMinSpacing with endToEndSpacing, vertical, prl, and exactAligned
- Example 3: trimMinSpacing with endToEndSpacing, vertical, prl, exactAligned, and edgeAligned
- Example 4: trimMinSpacing with endToEndSpacing, vertical, prl, exceptWidth, and layer
- Example 5: trimMinSpacing with prlSpacing, endToEndSpacing, horizontal, and insideLayers

Example 1: trimMinSpacing with prlSpacing, endToEndSpacing, and vertical

The width of TrimMetal1 shapes must be equal to 0.2 and their length must be less than or equal to 0.6. The shapes must extend by half of the required spacing of the wires (midTrack).

If the parallel run length between two shapes is zero, the spacing between them must be greater than or equal to 0.3; but if the parallel run length 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.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

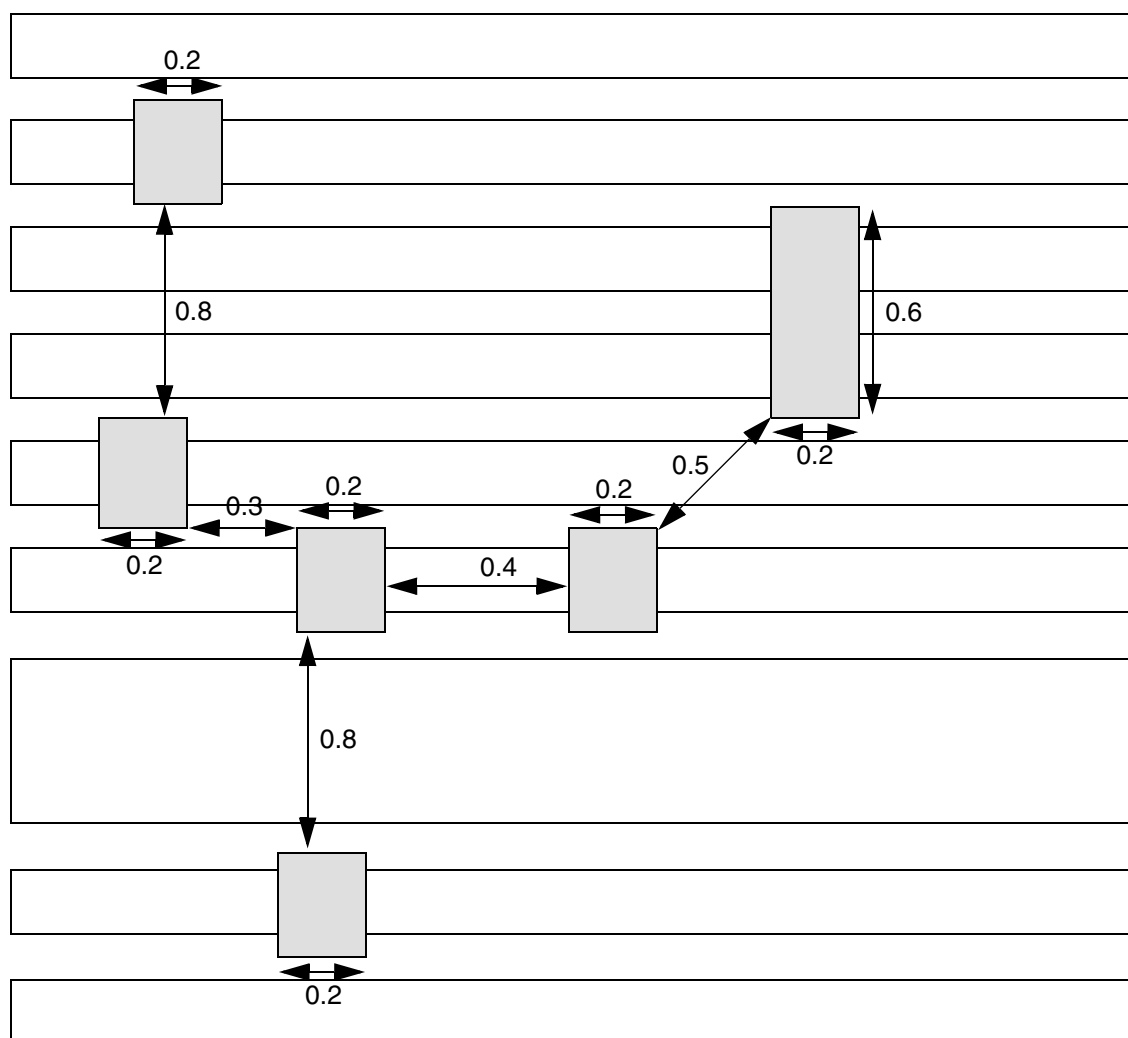
The end-to-end spacing in the vertical direction must be greater than or equal to 0.8 if the parallel run length between the shapes is greater than zero.

```
spacings(  
  ( trimShape "TrimMetal1"  
    'exactWidth 0.2  
    'maxLength 0.6  
    "midTrack"  
  )  
  ( trimMinSpacing "TrimMetal1"  
    'prlSpacing (0.3 0.4)  
    'endToEndSpacing 0.8  
    'vertical  
    0.5  
  )  
) ;spacings
```

 TrimMetal1

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)



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, vertical, prl, and exactAligned

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 exactly 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.

Virtuoso Technology Data Constraints Reference

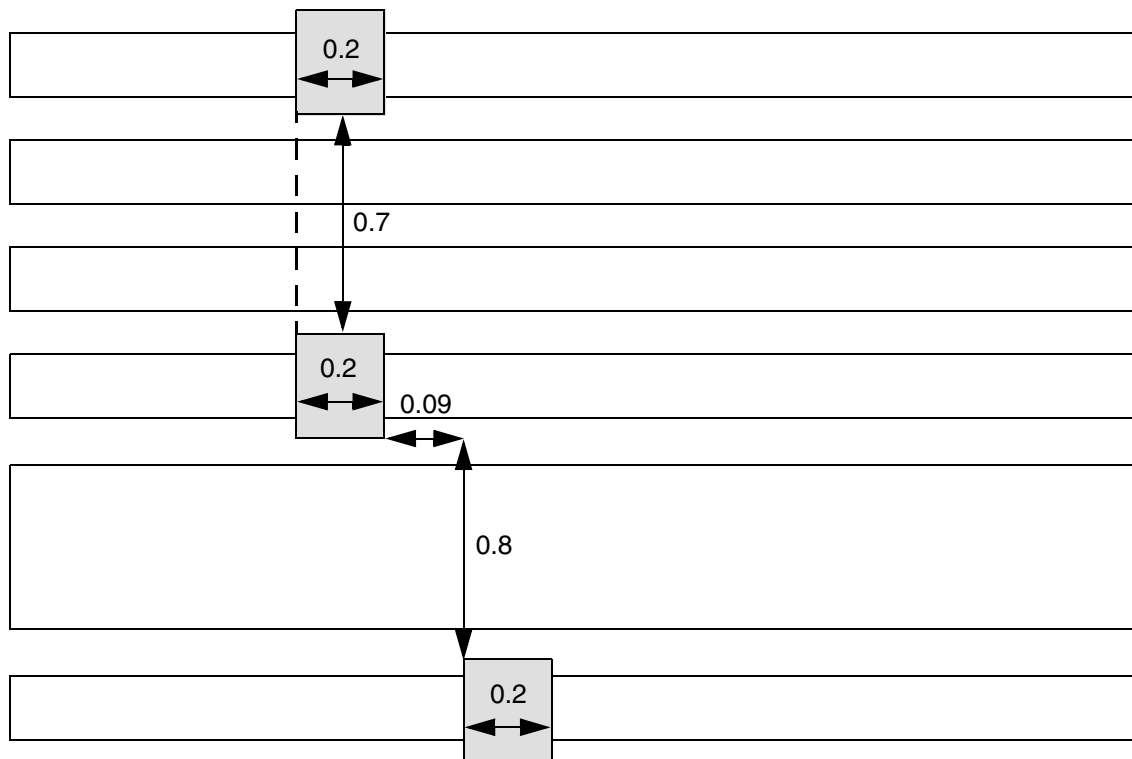
Spacing Constraints (One Layer)

```

spacings(
  ( trimMinSpacing "TrimMetal1"
    'endToEndSpacing 0.8
    'vertical
    'prl -0.1
    'exactAligned 0.7
    0.5
  )
) ;spacings

```

□ TrimMetal1



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 exactly aligned is 0.7.

Example 3: trimMinSpacing with endToEndSpacing, vertical, prl, exactAligned, and edgeAligned

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 exactly aligned, the end-to-end spacing in the vertical direction must be greater than or equal to 0.7, and if the left edge of a shape is exactly aligned with the right edge of another shape, the end-to-end

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

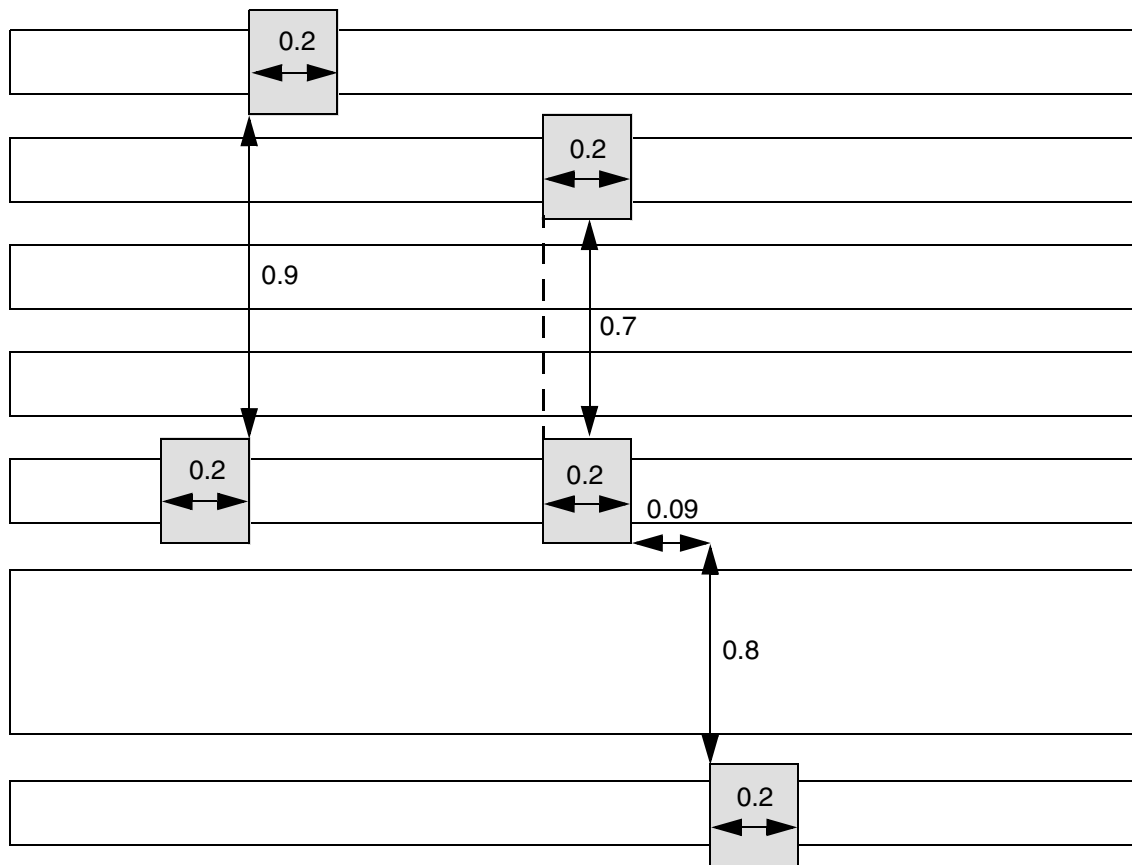
spacing between the shapes in the vertical direction must be greater than or equal to 0.9. Otherwise, the spacing between the shapes must be at least 0.5.

```

spacings(
  ( trimMinSpacing "TrimMetal1"
    'endToEndSpacing 0.8
    'vertical
    'prl -0.1
    'exactAligned 0.7
    'edgeAligned 0.9
    0.5
  )
) ;spacings

```

TrimMetal1



PASS. The vertical end-to-end spacing between shapes with parallel run length equal to 0.09 (<0.1) is 0.8; the vertical end-to-end spacing between shapes that are exactly aligned is 0.7; and the vertical end-to-end spacing between the shape that has its left edge exactly aligned with the right edge of another shape is 0.9.

Virtuoso Technology Data Constraints Reference

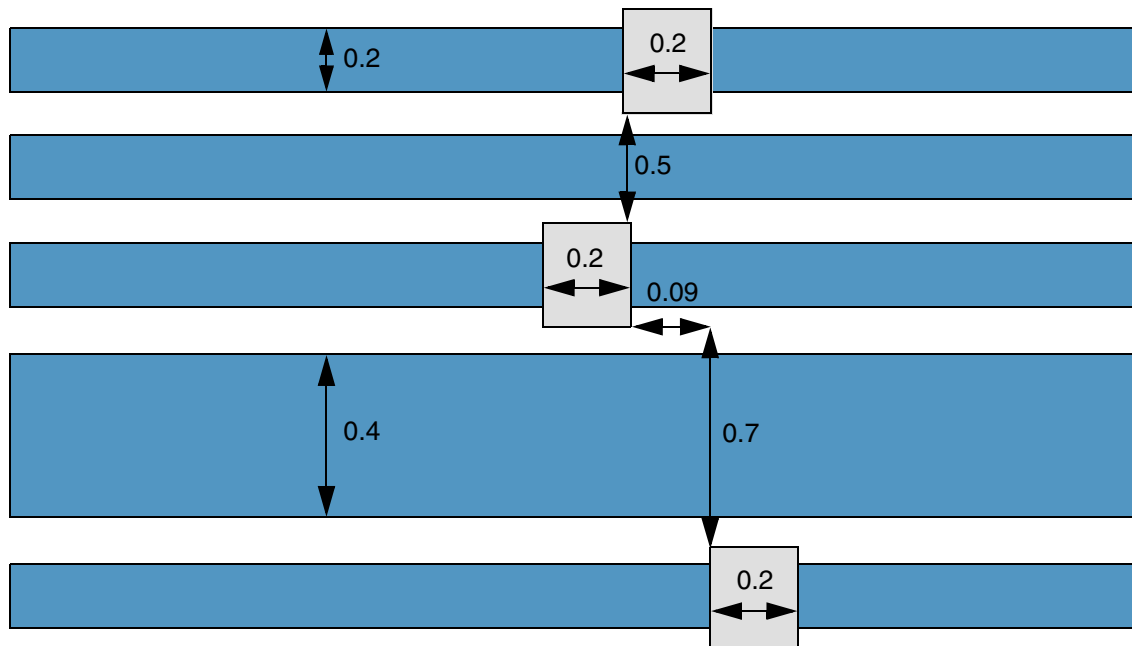
Spacing Constraints (One Layer)

Example 4: trimMinSpacing with endToEndSpacing, vertical, prl, exceptWidth, and layer

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.

```
spacings(  
  ( trimMinSpacing "TrimMetal1"  
    'endToEndSpacing 0.8  
    'vertical  
    'prl -0.1  
    'exceptWidth 0.4 'layer "Metal1"  
    0.5  
  )  
) ;spacings
```

□ TrimMetal1
■ Metal1



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.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example 5: trimMinSpacing with prlSpacing, endToEndSpacing, horizontal, and insideLayers

If the parallel run length between two shapes is zero, the spacing between them must be greater than or equal to 0.3; but if the parallel run length 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 0.51. The end-to-end spacing in the horizontal direction must be greater than or equal to 0.8 if the parallel run length between the shapes is greater than zero.

The constraint applies only if the shapes on layer NP overlap the shapes on layer Metal1 or Metal2.

```
spacings(  
  ( trimMinSpacing "NP"  
    'prlSpacing (0.3 0.4)  
    'endToEndSpacing 0.8  
    'horizontal  
    'insideLayers ("Metal2" "Metal3")  
    0.51  
  )  
) ;spacings
```

viaKeepoutZone

```

spacings (
    ( viaKeepoutZone tx_cutLayer
        'cutClass {f_width | (f_width f_length) | t_name}
        ['otherCutClass {f_width1 | (f_width1 f_length1) | t_name1}]
        'dualExtension (f_ext1 f_ext2)
        ['dualSideExtension (f_sideExt1 f_sideExt2) ['sideVertical]]
        ['sameMask]
        ['exactAligned f_spacing
            ['allEdge | 'shortEdge | 'longEdge]
        ]
        ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
            ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
            |
            ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
                ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        ]
        ['cornerOnly]
        ['exceptSameNet]
        ['viasAnyConnectivity | 'viasOnSameMetal | 'viasOnDiffMetal]
        f_extension
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Defines a keepout zone that determines the spacing between via cuts of the specified cut class and any other cut class, or between via cuts of the two specified cut classes. The keepout zone is a combination of the keepout zones defined by the constraint value (*extension*) and by the parameters 'dualExtension (can be specified for both square and rectangular cut classes) and, optionally, 'dualSideExtension (can be specified only for rectangular cut classes). If the optional 'sideVertical parameter is specified, 'dualSideExtension, which is typically specified only for rectangular cut classes, can be specified for square cut classes as well.

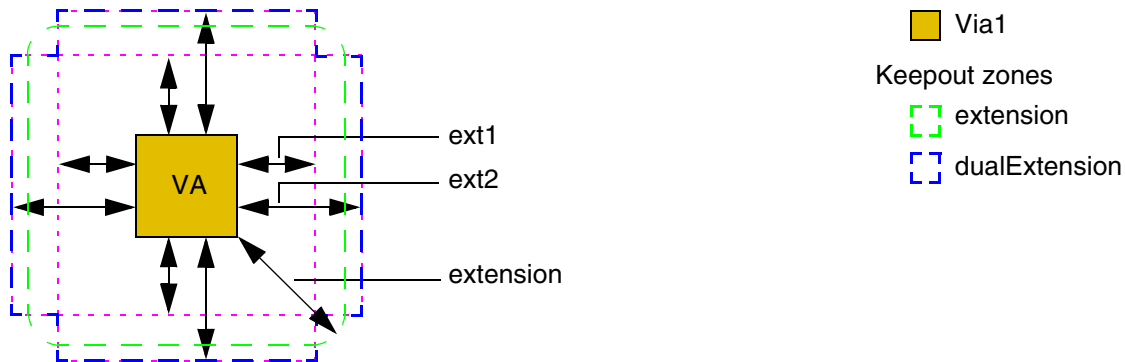
1. When 'sideVertical and 'cornerOnly are NOT specified

The following figure illustrates the two keepout zones for a square via cut, one defined by the constraint value (green) and the other defined by 'dualExtension (blue). When only 'dualExtension is specified for a square or rectangular via cut, the keepout zone is the

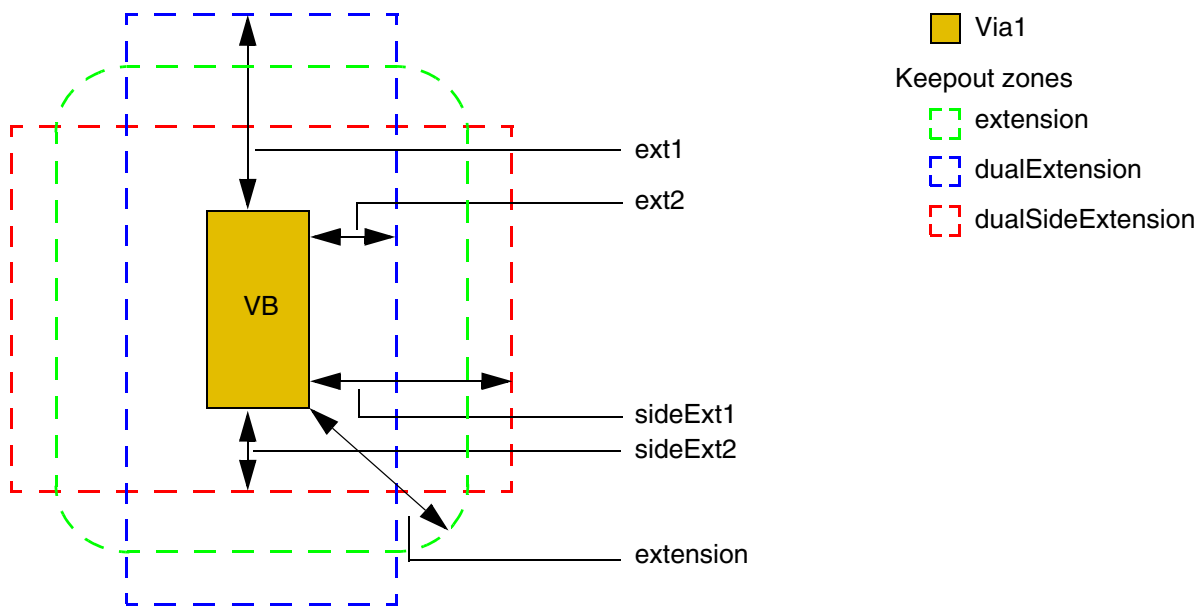
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

union of the two regions formed by extending outward the two extension values, *ext1* and *ext2*, from each pair of opposite edges of the via cut.



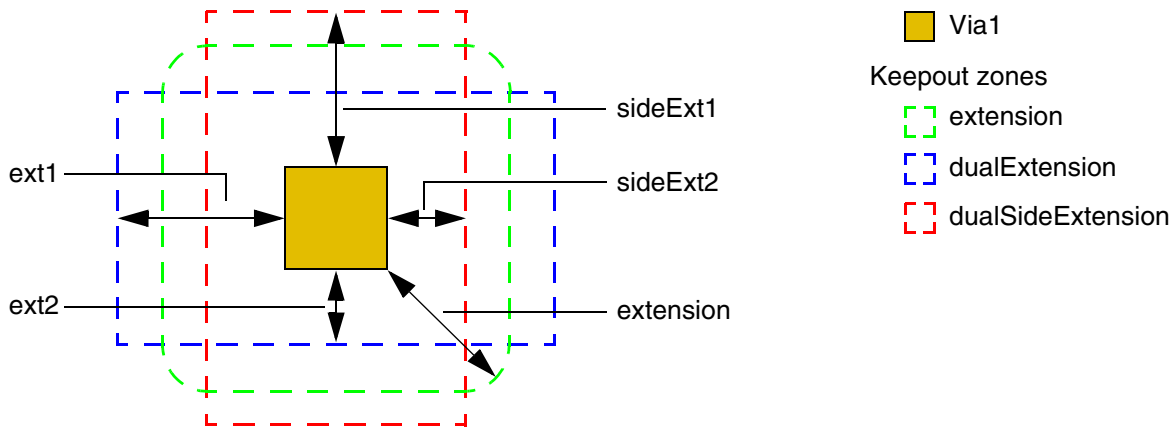
The following figure illustrates the three keepout zones for a rectangular via cut when both 'dualExtension' and 'dualSideExtension' are specified: the first defined by the constraint value (green), the second defined by 'dualExtension' (blue, with *ext1* on the ends and *ext2* on the sides), and the third defined by 'dualSideExtension' (red, with *sideExt1* on the sides and *sideExt2* on the ends).



A violation occurs if a via cut of any cut class or of the type specified using 'otherCutClass overlaps any of the keepout zones.

2. When 'sideVertical is specified

When 'sideVertical is specified, both 'dualExtension and 'dualSideExtension can be specified for a square via cut. In this setting, 'dualExtension is used to define a horizontal keepout zone (blue) and 'dualSideExtension is used to define a vertical keepout zone (red), as shown in the following figure.



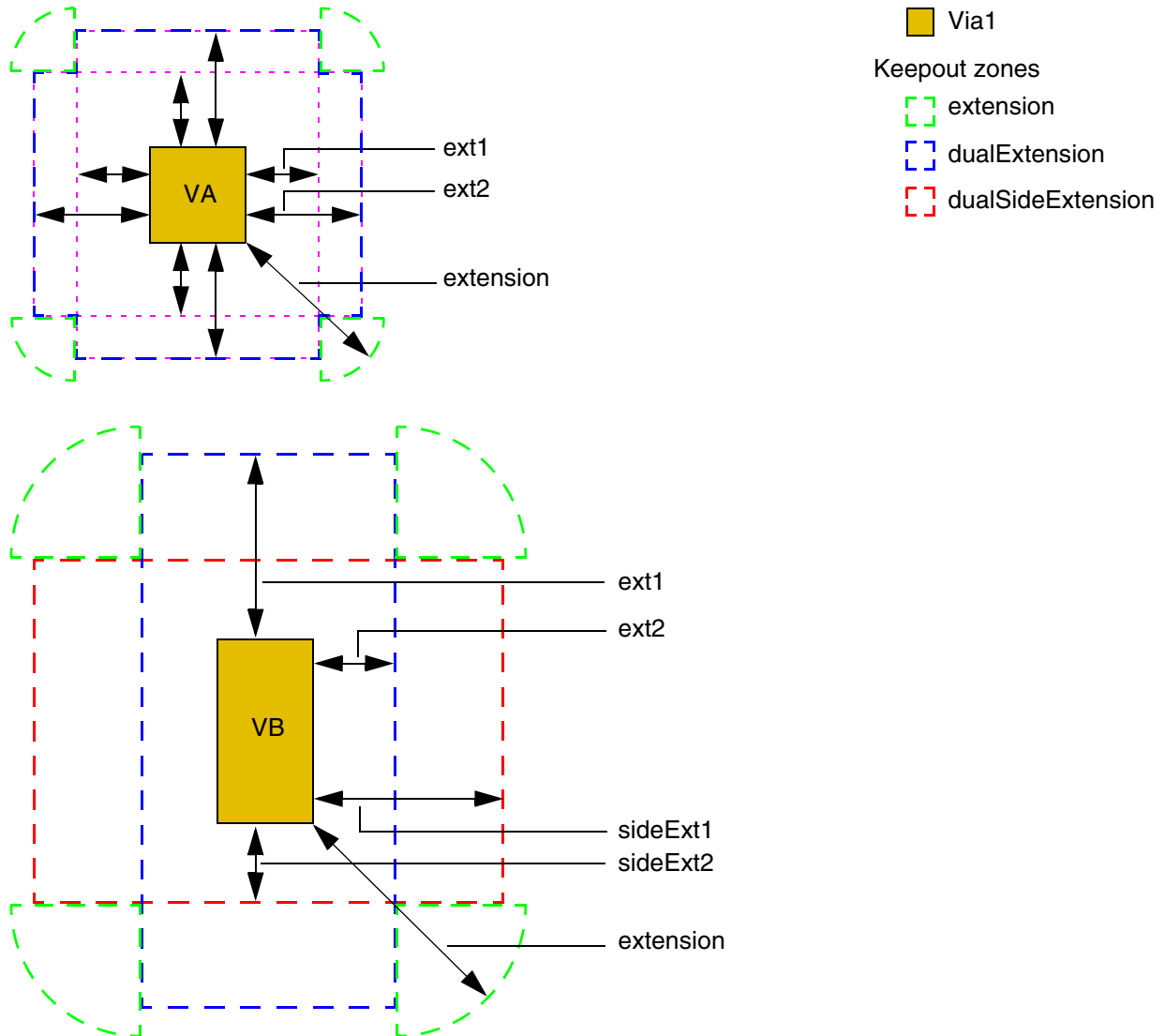
3. When 'cornerOnly is specified

When 'cornerOnly is specified, all cut edges are extended outward by *extension* in Euclidean measurement to form the keepout zone for the constraint value. However, this

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

keepout zone applies only at the corners that are outside the keepout zones defined by 'dualExtension and 'dualSideExtension, as shown in the following figure.



Values

tx_cutLayer

The cut layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_extension

All cut edges are extended outward by this value in Euclidean measurement to define a keepout zone.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'otherCutClass {f_width1 | (f_width1 f_length1) | t_name1}`

The cut class that must not overlap any keepout zone, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'dualExtension (f_ext1 f_ext2)`

The two values together define a second keepout zone.

- For a square or rectangular via cut when `'dualSideExtension` is not specified:
The keepout zone is the union of the two regions formed by extending outward the two extension values, `ext1` and `ext2`, from each pair of opposite edges of the via cut.
- For a rectangular via cut when `'dualSideExtension` is also specified:
The keepout zone extends outward by `ext1` from both ends (short edges) of the via cut and by `ext2` from both sides (long edges) of the via cut.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'dualSideExtension (f_sideExt1 f_sideExt2)`

The two values together define a third keepout zone that extends outward by *sideExt1* from both sides and by *sideExt2* from both ends of the rectangular via cut.

Note: If `'sideVertical` is specified, this parameter can be set for both square and rectangular via cuts.

`'sideVertical`

Both `'dualExtension` and `'dualSideExtension` parameters can be specified for square cut classes, with `'dualExtension` defining a horizontal keepout zone and `'dualSideExtension` defining a vertical keepout zone.

Type: Boolean

`'sameMask`

The constraint applies only to via cuts on the same mask.

Type: Boolean

`'exactAligned f_spacing`

The distance between exactly aligned via cuts must be greater than or equal to this value. The via cuts are not subject to the keepout zone check.

`'allEdge | 'shortEdge | 'longEdge`

The edge on which rectangular via cuts must be exactly aligned to trigger the `'exactAligned` check.

- `'allEdge`: (Default) The rectangular via cuts can be exactly aligned on any edge.
- `'shortEdge`: The rectangular via cuts must be exactly aligned on the end/short edges.
- `'longEdge`: The rectangular via cuts must be exactly aligned on the side/long edges.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*cutLayer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*cutLayer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'cornerOnly`

The keepout zone defined by the constraint value is applied only on the corners that are not covered by the keepout zones defined by `'dualExtension` and `'dualSideExtension`.

Type: Boolean

`'exceptSameNet`

Specifies that the constraint does not apply to via cuts on the same net.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

`'viasAnyConnectivity` | `'viasOnSameMetal` | `'viasOnDiffMetal`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies to vias of the specified connectivity type.

- `'viasAnyConnectivity`: The constraint applies to all cut shapes, regardless of the connectivity type.
- `'viasOnSameMetal`: The constraint applies to a subset of cut shapes that are electrically connected by a contiguous metal on either the layer below or the layer above.
- `'viasOnDiffMetal`: The constraint applies to a subset of cut shapes that are electrically connected by different metals on the layer below and the layer above.

This parameter may be specified only if `exceptSameNet` is not specified.

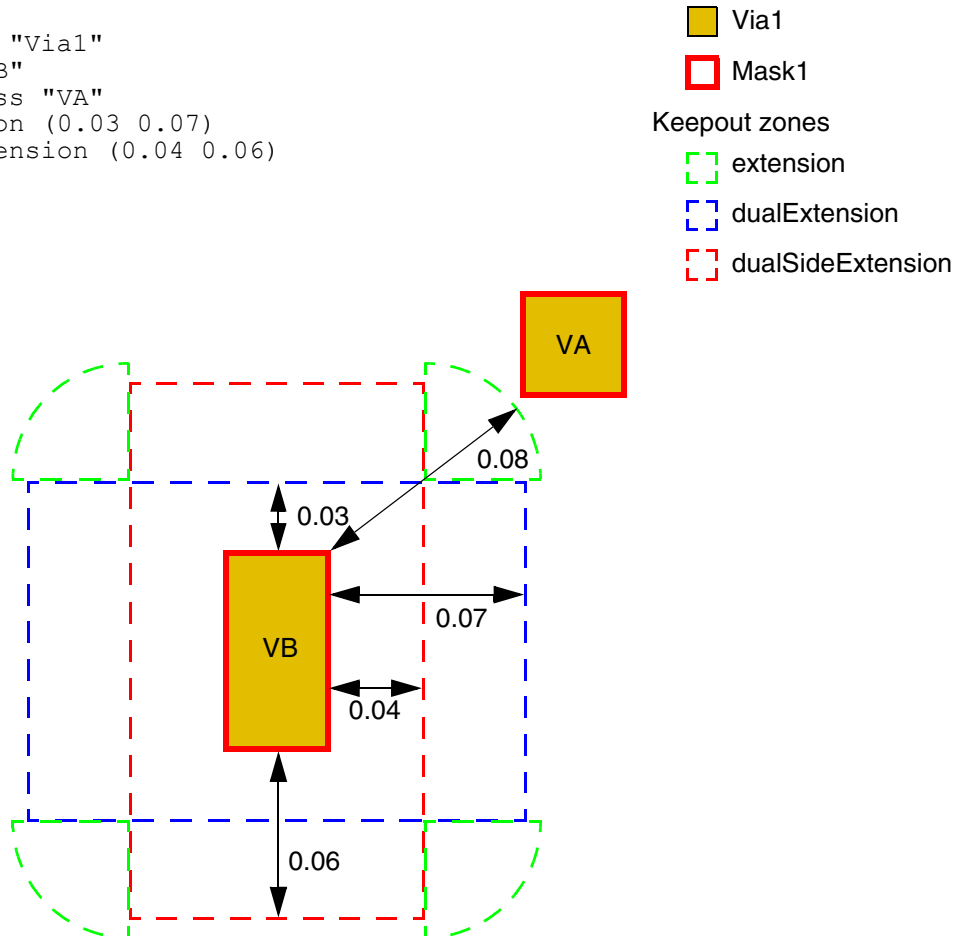
Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Example

On layer Via1, a VA via cut must not overlap with the keepout zones defined for a VB via cut if both via cuts are on the same mask. Additionally, the keepout zone defined by the constraint value 0.08 applies only on the corners where the keepout zones defined by 'dualExtension and 'dualSideExtension do not apply.

```
spacings(  
  ( viaKeepoutZone "Via1"  
    'cutClass "VB"  
    'otherCutClass "VA"  
    'dualExtension (0.03 0.07)  
    'dualSideExtension (0.04 0.06)  
    'sameMask  
    'cornerOnly  
    0.08  
  )  
) ;spacings
```



voltageLabelMapping

```
spacings(  
    ( voltageLabelMapping tx_layer  
        'layer t_layer  
        'purpose t_purpose  
        [t_voltage]  
    )  
) ;spacings
```

Specifies the layer and purpose for which voltage labels will be created.

Values

<i>tx_layer</i>	The voltage label is created on a shape on this layer on the net that is being labeled.
<i>t_voltage</i>	A label is created for nets based on the value specified for voltage. The valid values are <code>maxVoltage</code> and <code>minVoltage</code> . If it is not specified, labels are created for all voltages.

Parameters

<code>'layer t_layer</code>	The layer on which the label will be created.
<code>'purpose t_purpose</code>	The purpose on which the label will be created.

Example

Labels showing minimum voltages on nets will be drawn on the layer-purpose pair Metal1 m1min.

```
spacings(  
    ( voltageLabelMapping "Metal1"  
        'layer "Metal1"  
        'purpose "m1min"  
        "minVoltage"  
    )  
) ;spacings
```


voltageMarkerMapping

```
spacings(  
    ( voltageMarkerMapping tx_layer  
      'layer t_layer  
      'purpose t_purpose  
      f_voltage  
    )  
); spacings
```

Specifies the layer and purpose for which voltage markers will be created.

Values

<i>tx_layer</i>	The voltage marker is created on a shape on this layer on the net that is being labeled.
<i>f_voltage</i>	A marker is created for nets with the specified voltage.

Parameters

<code>'layer t_layer</code>	The layer on which the marker will be created.
<code>'purpose t_purpose</code>	The purpose on which the marker will be created.

Example

Markers showing the specified voltages will be drawn on nets on the layer-purpose pair Metal1 m1min.

```
spacings(  
    ( voltageMarkerMapping "Metal1"  
      'layer "Metal1"  
      'purpose "m1min"  
      1.2  
    )  
); spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (One Layer)

Spacing Constraints (Two Layers)

This topic includes the following constraints:

- [allowedSpacingRanges \(Two layers\)](#)
- [forbiddenExtensionRanges](#)
- [gateSpacingRanges](#)
- [maxSpacing](#)
- [minCenterLineSpacing \(Two layers\)](#)
- [minClusterSpacing \(Two layers\)](#)
- [minCornerSpacing \(Two layers\)](#)
- [minCutEdgeSpacing](#)
- [minCutRoutingSpacing \(Two layers\)](#)
- [minInnerVertexSpacing \(Two layers\)](#)
- [minInnerVertexSpacing \(Three layers\)](#)
- [minNeighboringShapesSpacing](#)
- [minNeighborVoltageSpacing \(Two layers\)](#)
- [minOppositeClearance \(Two layers\)](#)
- [minSameNetSpacing \(Two layers\)](#)
- [minSideSpacing \(Two layers\)](#)
- [minSpacing \(Two layers\)](#)
- [minSpacingOver](#)
- [minTouchingDirSpacing](#)
- [minVoltageSpacing \(Two layers\)](#)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

- shapeRequiredSpacing
- trimMinCutSpacing
- trimMinSpacing (Two layers)

allowedSpacingRanges (Two layers)

```

spacings (
    ( allowedSpacingRanges tx_layer1 tx_layer2
      'width f_width
      (g_ranges)
    )
) ;spacings

spacingTables (
    ( allowedSpacingRanges tx_layer1 tx_layer2
      (( "width" nil nil ["width" nil nil] )
        ['width f_width]
        [f_default]
      )
      (g_table)
    )
) ;spacingTables

```

Defines the spacing required between shapes on two different layers.

The spacing can optionally depend on the width of the shapes. Multiple allowed ranges and discrete values can be specified.

In some processes, it is necessary to insert assist or scattering bars between minimum-sized gates and other poly shapes to ensure that the gates print on the mask correctly. This constraint defines the spacing required between gates below the specified minimum width and the shapes on the specified second layer. The allowable spacing can depend on the width of one or both shapes.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The allowed spacing ranges are specified as follows: (spacingRange1 spacingRange2 ...) Type: Floating-point values specifying the spacing <u>ranges</u> that are allowed

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

"width" nil nil ["width" nil nil]

This identifies the index for *table*.

g_table

The format of a 1-D *table* row is as follows:

(f_width f_spacing)

where, *f_width* is the width of the wider of the two shapes and *f_spacing* is the spacing allowed when width is greater than or equal to the corresponding index.

The format of a 2-D table is as follows:

((f_width1 f_width2) f_spacing)

where, *f_width1* and *f_width2* are the widths of the two shapes and *f_spacing* is the spacing allowed when both widths are greater than or equal to the corresponding indexes.

For each table entry, the last spacing range must be open to the right.

Type: A 1-D or 2-D table specifying floating-point width and spacing values

Parameters

'width *f_width*

The constraint applies only if the width of both shapes is less than this value.

f_default

The spacing value to be used when no table entry applies.

Example 1

The spacing between a shape on Gate1 and a shape on Poly1 must be either of the following:

- greater than 0.3 and less than or equal to 0.5, or
- greater than or equal to 0.8.

The spacing between a shape on Gate2 and a shape on Poly2 must be either of the following:

- greater than 0.2 and less than 0.4, or
- greater than or equal to 0.8.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

```
spacings (
  ( allowedSpacingRanges "Gate1" "Poly1"
    ("(0.3 0.5]" ">=0.8")
  )
  ( allowedSpacingRanges "Gate2" "Poly2"
    ("(0.2 0.4)" ">=0.8")
  )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example 2

The spacing between a shape on Poly and a shape on Active is determined as follows:

- Must be equal to 0.12 or greater than or equal to 0.15 if the width of the wider shape is greater than or equal to 0.1.
- Must be equal to 0.15; or greater than or equal to 0.2 and less than or equal to 0.21; or greater than or equal to 0.3 if the width of the wider shape is greater than or equal to 0.2.

```
spacingTables(  
  ( allowedSpacingRanges "Poly" "Active"  
    (( "width" nil nil ))  
    (  
      0.1 (0.12 ">=0.15")  
      0.2 (0.15 "[0.2 0.21]" ">=0.3")  
    )  
  )  
) ;spacingTables
```


Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

forbiddenExtensionRanges

```
orderedSpacings (
  ( forbiddenExtensionRanges  tx_layer1  tx_layer2
    ['topExtensionRange      g_topExtensionRange]
    ['bottomExtensionRange    g_bottomExtensionRange]
    ['leftExtensionRange      g_leftExtensionRange ]
    ['rightExtensionRange     g_rightExtensionRange ]
  )
) ;orderedSpacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies forbidden enclosures of one layer on another layer with asymmetric values for top, bottom, left, and right sides.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Parameters

`'topExtensionRange g_topExtensionRange`

The constraint applies only if the enclosure of *layer1* on the top side edge of the *layer2* is in this range.

Type: Floating-point values specifying width ranges

`'bottomExtensionRange g_bottomExtensionRange`

The constraint applies only if the enclosure of *layer1* on the bottom side edge of the *layer2* is in this range.

Type: Floating-point values specifying width ranges

`'leftExtensionRange g_leftExtensionRange`

The constraint applies only if the enclosure of *layer1* on the left side edge of the *layer2* is in this range.

Type: Floating-point values specifying width ranges

`'rightExtensionRange g_rightExtensionRange`

The constraint applies only if the enclosure of *layer1* on the right side edge of the *layer2* is in this range.

Type: Floating-point values specifying width ranges

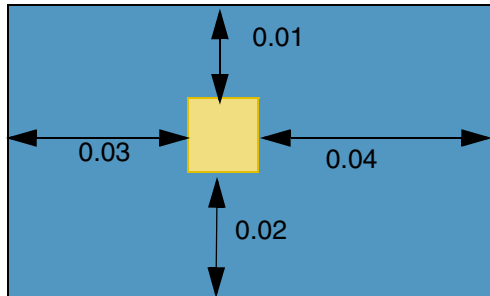
Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example

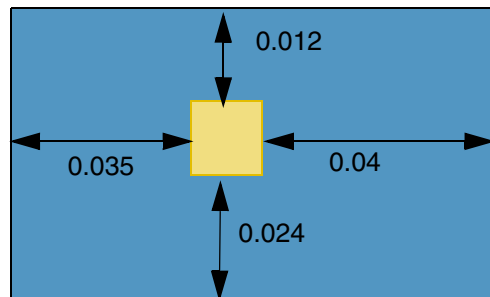
```
orderedSpacings (
  (forbiddenExtensionRanges
    'topExtensionRange      "Nwell"  "Oxide"
    'bottomExtensionRange   "[0.02 0.03]"
    'leftExtensionRange      "[0.03 0.04]"
    'rightExtensionRange    "[0.04 0.05]"
    'rightExtensionRange    "[0.05 0.06]"
  ) ;orderedSpacings
```

■ Oxide
■ Nwell



PASS.

```
orderedSpacings (
  (forbiddenExtensionRanges
    'topExtensionRange      "Nwell"  "Oxide"
    'bottomExtensionRange   "[0.01 0.02]"
    'leftExtensionRange      "[0.02 0.03]"
    'rightExtensionRange    "[0.03 0.04]"
    'rightExtensionRange    "[0.04 0.05]"
  ) ;orderedSpacings
```



FAIL.

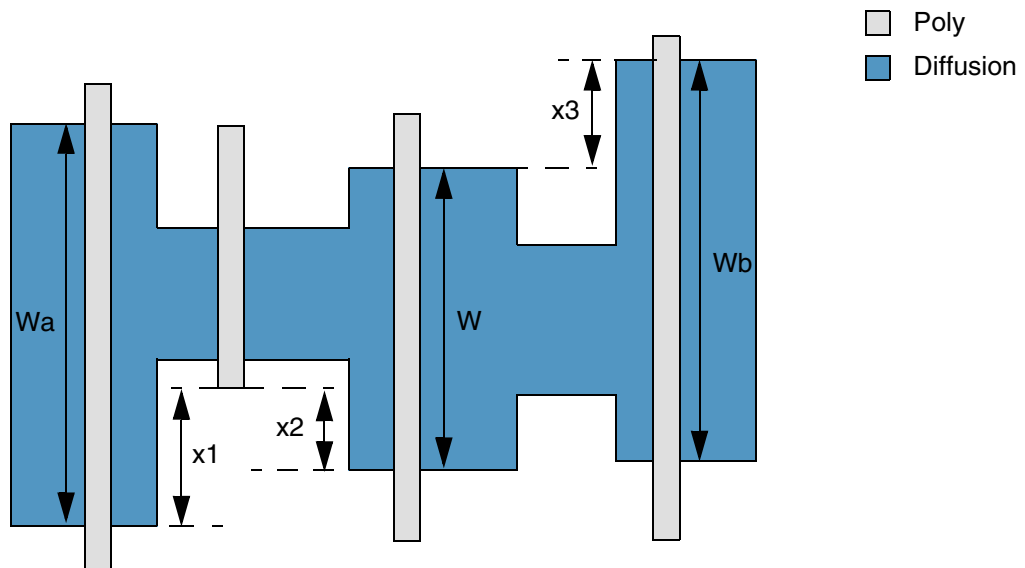
gateSpacingRanges

```
spacings (  
    ( gateSpacingRanges tx_layer  
      'ratio f_ratio  
      'length f_length  
      (g_ranges)  
    )  
) ;spacings
```

Defines the spacing between a gate and neighboring poly shapes.

The constraint applies only when the width of the gate (or channel length) is less than or equal to *length*. However, if the ratio of the distance between the top of the gate and a neighboring poly shape to the width of the gate is less than *ratio*, the constraint does not apply.

In the example below, the gate with width *W* has to its left a poly extension shape and the distance of this poly extension from the top of the gate with width *W* is *x2*. If $x2/W$ is less than *ratio*, the constraint does not apply. Similarly, if $x1/Wa$ or $x3/Wb$ is less than *ratio*, the constraint does not apply.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The allowed spacing ranges between a gate and the neighboring poly shapes. Type: A list of floating-point value <u>ranges</u> specifying allowed spacing.

Parameters

<i>'ratio f_ratio</i>	The constraint does not apply if the ratio of the distance between the top of the gate and a neighboring poly shape to the width of the gate is less than this value.
<i>'length f_length</i>	The constraint applies only if the channel length of the gate is less than or equal to this value.

Example

The spacing between a gate and a neighboring Poly shape must be greater than zero or less than 0.5 or greater than or equal to 0.1 if the width of the gate is less than or equal to 2 and the ratio of the distance between the top of the gate and a neighboring poly shape to the width of the gate is less than 0.25.

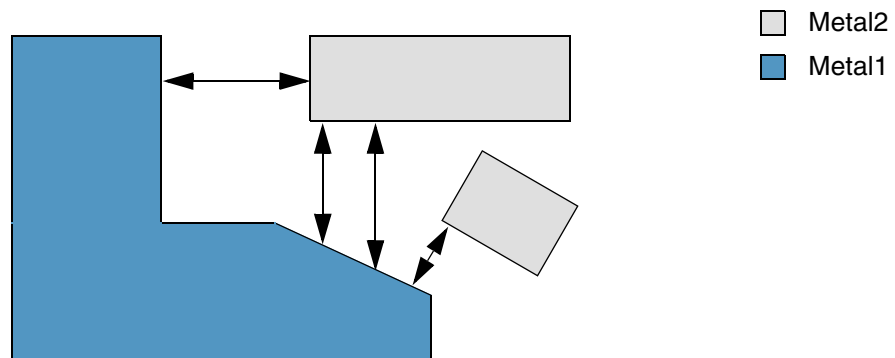
```
spacings (
  ( gateSpacingRanges "Poly"
    'ratio 0.25
    'length 2
    ("(0 0.5)" ">= 1")
  )
) ;spacings
```

maxSpacing

```
spacings(  
    ( maxSpacing tx_layer1 tx_layer2  
      f_spacing  
    )  
) ;spacings
```

Defines the maximum spacing between shapes on the specified layers.

The constraint applies only to non-intersecting shapes on two different layers.



Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The maximum spacing between the shapes.

Parameters

None

Example

The maximum spacing between a Metal1 shape and a Metal2 shape must be 4.6.

```
spacings(  
    ( maxSpacing "Metal1" "Metal2"  
      4.6  
    )  
) ;spacings
```

minCenterLineSpacing (Two layers)

```
orderedSpacings (
    ( minCenterLineSpacing tx_layer1 tx_layer2
      f_spacing
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between the centerlines of the shapes on the specified layers.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing from the centerline of the <i>layer1</i> shape to the <i>layer2</i> shape must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

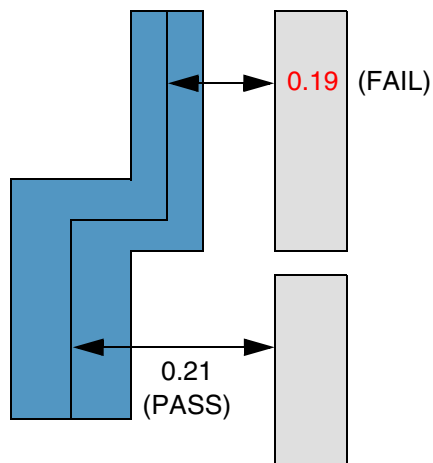
Spacing Constraints (Two Layers)

Example

The spacing between the centerlines of a Metal1 shape and a Metal2 shape must be greater than or equal to 0.2.

```
orderedSpacings(  
  ( minCenterLineSpacing "Metal1" "Metal2"  
    0.2  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



minClusterSpacing (Two layers)

```
orderedSpacings (
    ( minClusterSpacing tx_layer1 tx_layer2
      'widthRange g_widthRange
      'numShapesRange g_numShapesRange
      'spacingRange g_spacingRange
      ['centerLineDistance]
      ['horizontal | 'vertical]
      ['otherWidthRange g_otherWidthRange]
      ['groupHorizontal | 'groupVertical]
      ['prl f_prl]
      g_spacing
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between two neighboring clusters or a cluster and other neighboring shapes that may or may not be part of another cluster, on the two specified layers. The shapes in a cluster must be exactly aligned and must satisfy all of the following conditions:

- The width of each shape in the cluster must be in the specified width range, *widthRange*.
- The edge-to-edge spacing or the spacing between the centerlines of neighboring shapes in the cluster must be in the specified spacing range, *spacingRange*.
- The number of shapes that satisfy the two conditions listed above must be in the specified number range, *numShapesRange*.

Optionally, each cluster can be aligned either in the vertical direction or in the horizontal direction. The constraint can also be selectively applied to neighboring non-cluster shapes based on their widths.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

f_spacing

The spacing between a cluster and a neighboring shape must be greater than or equal to this value.

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.

Parameters

'widthRange *g_widthRange*

The widths of the shapes in a cluster must fall in this range.

Type: Floating-point values specifying a range of widths that trigger the constraint

'numShapesRange *g_numShapesRange*

The constraint applies only if the number of shapes in a cluster falls in this range.

Type: Integer values specifying the minimum and maximum number of shapes that a cluster can contain

'spacingRange *g_spacingRange*

This spacing between the centerlines of the shapes in a cluster must fall in this range.

Type: Floating-point values specifying a range of spacings that trigger the constraint

'centerLineDistance

The spacing is measured from the centerline of the cluster to centerline of the neighboring non-cluster shapes.

Note: The intra-cluster spacing is always measured from the centerline, irrespective of whether this parameter is specified.

Type: Boolean

'horizontal | 'vertical

The direction in which spacing is measured. If direction is not specified, spacing is measured in any direction.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

`'otherWidthRange g_otherWidthRange`

The constraint applies to a non-cluster shape only if its width falls in this range.

Type: Floating-point values specifying a range of widths that trigger the constraint

`'groupHorizontal | 'groupVertical`

The constraint applies only if a cluster is formed in the specified direction. If the direction is not specified, the cluster can be formed in either direction.

Type: Boolean

`'prl f_prl`

The constraint applies only if the parallel run length between the shapes in a cluster and a neighboring non-cluster shape is greater than or equal to this value.

Examples

- Example 1: minClusterSpacing with widthRange, numShapesRange, spacingRange, and otherWidthRange
- Example 2: minClusterSpacing with widthRange, numShapesRange, spacingRange, centerLineDistance, and otherWidthRange
- Example 3: minClusterSpacing with widthRange, numShapesRange, spacingRange, centerLineDistance, otherWidthRange, and groupVertical
- Example 4: minClusterSpacing with widthRange, numShapesRange, spacingRange, horizontal, otherWidthRange, and groupVertical

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example 1: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, and *otherWidthRange*

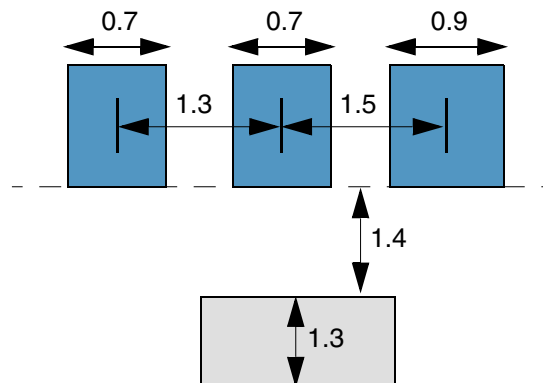
The spacing between a cluster and a neighboring Metal2 shape must be at least 1.4 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5.

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
orderedSpacings (  
  ( minClusterSpacing "Metal1" "Metal2"  
    'widthRange "[0.7 0.9]"  
    'numShapesRange "[3 4]"  
    'spacingRange "[1.3 1.5]"  
    'otherWidthRange "[1.2 1.5]"  
    1.4  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



PASS. All conditions for a cluster are satisfied: the number of shapes is 3 (which falls in the range [3 4]); the widths of the shapes fall in the range [0.7 0.9]; and the spacing measured between centerlines falls in the range [1.3 1.5]. The width of the non-cluster Metal2 shape is 1.3, which falls in the range [1.2 1.5], and the spacing between the shapes in the cluster and the non-cluster shape is 1.4.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example 2: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *centerLineDistance*, and *otherWidthRange*

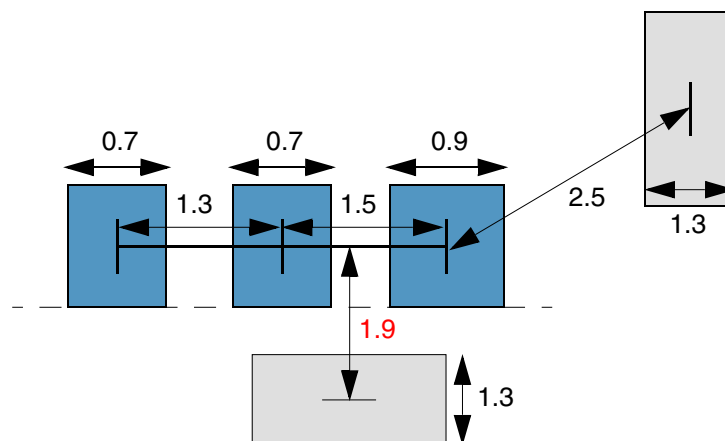
The spacing measured from the centerline of a cluster to the centerline of a neighboring Metal2 shape must be at least 2.0 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5.

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
orderedSpacings (  
  ( minClusterSpacing "Metal1" "Metal2"  
    'widthRange "[0.7 0.9]"  
    'numShapesRange "[3 4]"  
    'spacingRange "[1.3 1.5]"  
    'centerLineDistance  
    'otherWidthRange "[1.2 1.5]"  
    2.0  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



FAIL. All conditions for a cluster are satisfied and the width of the two neighboring shapes is 1.3 (falls in [1.2 1.5]). The spacing between the cluster centerline and the centerline of the neighboring Metal2 shape on the right is 2.5 (>2.0), but the spacing between the cluster centerline and the centerline of the neighboring Metal2 shape at the bottom is 1.9 (<2.0).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example 3: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *centerLineDistance*, *otherWidthRange*, and *groupVertical*

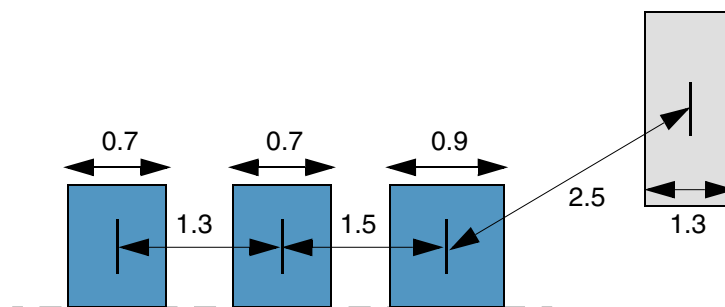
The spacing measured from the centerline of a vertical cluster to the centerline of a neighboring Metal2 shape must be at least 2.0 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5.

The shapes in the cluster must satisfy the following conditions:

- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
orderedSpacings (  
  ( minClusterSpacing "Metal1" "Metal2"  
    'widthRange "[0.7 0.9]"  
    'numShapesRange "[3 4]"  
    'spacingRange "[1.3 1.5]"  
    'centerLineDistance  
    'otherWidthRange "[1.2 1.5]"  
    'groupVertical  
    3.0  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



The constraint does not apply because the cluster direction is horizontal.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

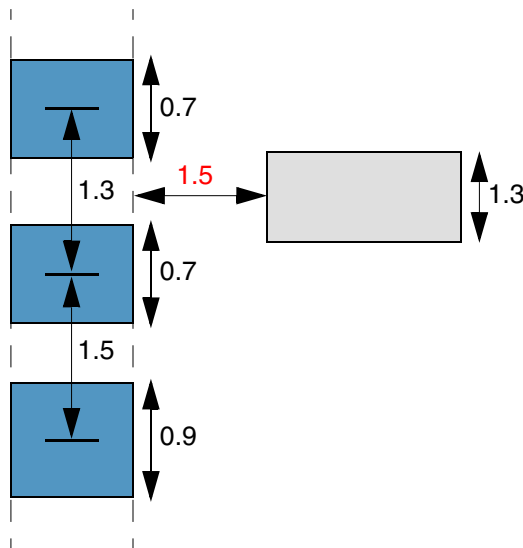
Example 4: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *horizontal*, *otherWidthRange*, and *groupVertical*

The horizontal spacing between a vertical cluster and a neighboring Metal2 shape must be at least 1.7 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5. The shapes in the cluster must satisfy the following conditions:

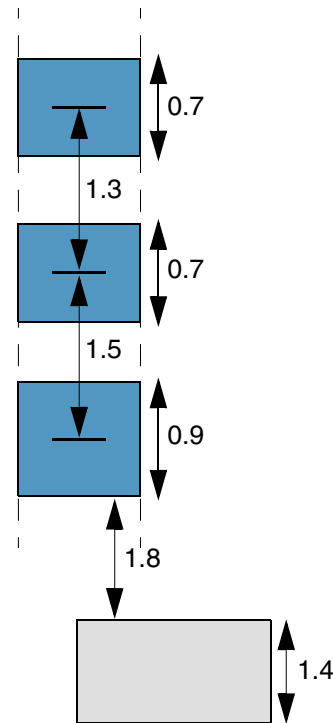
- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```
orderedSpacings (
  ( minClusterSpacing "Metal1" "Metal2"
    'widthRange "[0.7 0.9]"
    'numShapesRange "[3 4]"
    'spacingRange "[1.3 1.5]"
    'horizontal
    'otherWidthRange "[1.2 1.5]"
    'groupVertical
    1.7
  )
) ;orderedSpacings
```

□ Metal2
■ Metal1



a) FAIL. All conditions for a cluster are satisfied, the cluster direction is vertical, and the width of the non-cluster shape is 1.3 (falls in [1.2 1.5]). Additionally, the spacing between the shapes in the cluster and the neighboring non-cluster Metal2 shape is measured in the horizontal direction. However, the spacing is only 1.5 (<1.7).



b) The constraint does not apply because the spacing between the shapes in the cluster and the neighboring non-cluster shape is measured in the vertical direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

minCornerSpacing (Two layers)

```
orderedSpacings (
  ( minCornerSpacing tx_layer1 tx_layer2
    ['allCorner | 'concaveCorner | 'convexCorner]
    ['otherAllCorner | 'otherConcaveCorner | 'otherConvexCorner]
    ['deltaVoltage f_deltav]
    g_spacing
  )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing required between the corners of shapes on the specified layers.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_spacing</i>	The minimum spacing between the corners of shapes on the specified layers.

Parameters

'allCorner 'concaveCorner 'convexCorner	The constraint applies to corners of this type on <i>layer1</i> . Type: Boolean
'otherAllCorner 'otherConcaveCorner 'otherConvexCorner	The constraint applies to corners of this type on <i>layer2</i> . Type: Boolean
'deltaVoltage <i>f_deltav</i>	The constraint applies if the maximum voltage difference between the two shapes is greater than this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

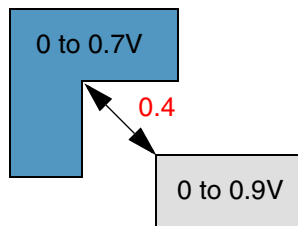
Example

The spacing between Metal1 and Metal2 corners must be greater than or equal to 0.5 if the following conditions are met:

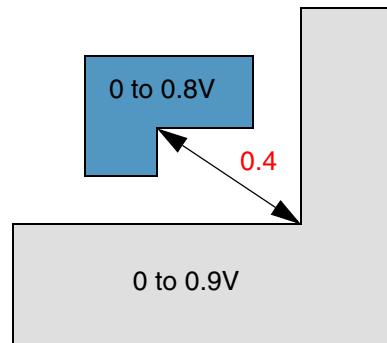
- The Metal1 corner is concave.
- The Metal2 corner is concave or convex.
- The maximum voltage difference between Metal1 and Metal2 shapes is greater than 0.8V.

```
orderedSpacings (  
  ( minCornerSpacing "Metal1" "Metal2"  
    'concaveCorner  
    'otherAllCorner  
    'deltaVoltage 0.8  
    0.5  
  )  
) ;orderedSpacings
```

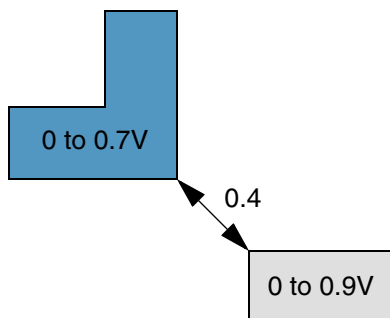
□ Metal2
■ Metal1



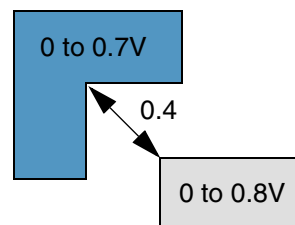
a) FAIL. The maximum voltage difference is 0.9 (>0.8), but the spacing is only 0.4 (<0.5).



b) FAIL. The maximum voltage difference is 0.9 (>0.8), but the spacing is only 0.4 (<0.5).



c) The constraint does not apply to Metal1 convex corners.



d) The constraint does not apply because the maximum voltage difference between the two shapes is 0.8 (and not greater than 0.8V).

Virtuoso Technology Data Constraints Reference

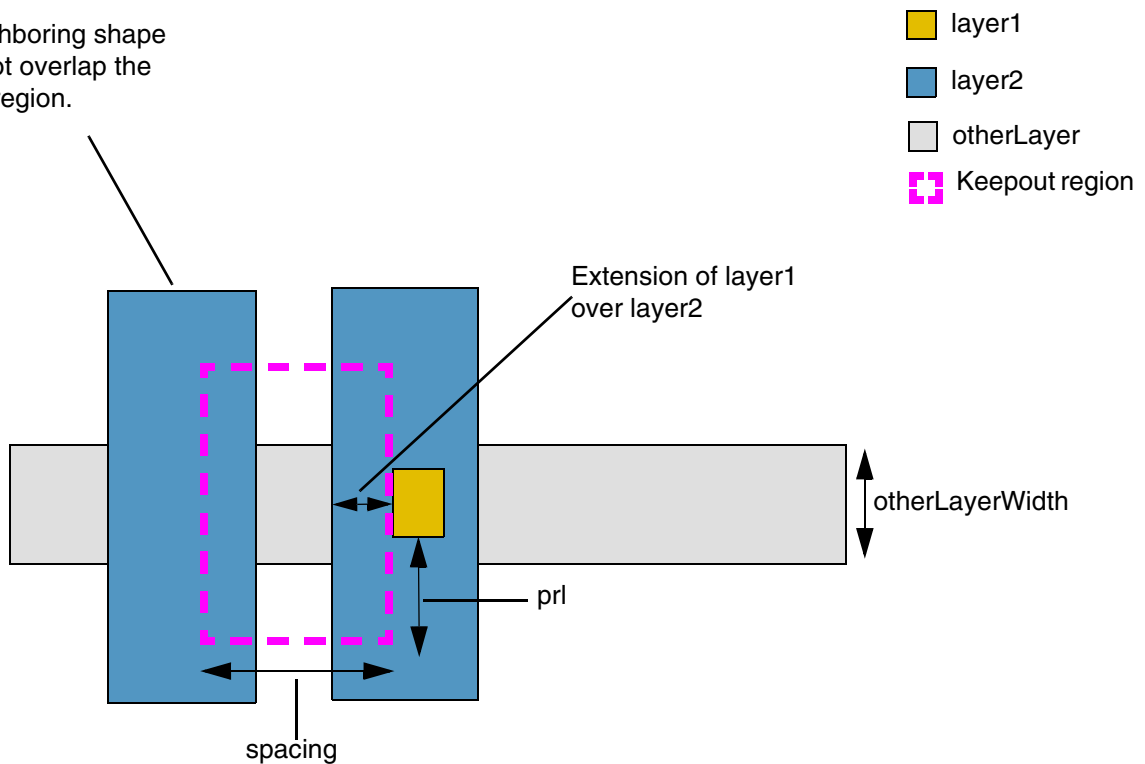
Spacing Constraints (Two Layers)

minCutEdgeSpacing

```
orderedSpacings (  
  ( minCutEdgeSpacing tx_layer1 tx_layer2  
    'prl f_prl  
    'exceptEdgeExtension f_exceptEdgeExtension  
    ['horizontal | 'vertical]  
    ['otherLayer tx_otherLayer  
    ['otherLayerWidth g_otherLayerWidthRange]  
  )  
  f_spacing  
)  
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum orthogonal spacing between a shape on a cut layer (*layer1*) and a shape on a metal layer (*layer2*).

This neighboring shape should not overlap the keepout region.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Values

<i>tx_layer1</i>	The first layer (cut layer) on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<i>tx_layer2</i>	The second layer (metal layer) on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<i>f_spacing</i>	The minimum required spacing.

Parameters

<i>'prl f_prl</i>	The constraint applies only if a neighboring shape on <i>layer2</i> has a parallel run length greater than the specified value from a shape on <i>layer1</i> . The direction of parallel run length is orthogonal to the spacing direction.
<i>'exceptEdgeExtension f_exceptEdgeExtension</i>	The spacing value of the constraint does not apply if the <i>layer1</i> edge facing the <i>layer2</i> shape has an extension by <i>layer2</i> that is greater than or equal to this parameter. The extension is measured in the spacing direction.
<i>'horizontal 'vertical</i>	The direction in which spacing is measured. If not specified, spacing is measured in both directions. Type: Boolean
<i>'otherLayer tx_otherLayer</i>	The constraint applies if the <i>layer1</i> shape intersects with or touches a shape on this layer.
<i>'otherLayerWidth g_otherLayerWidthRange</i>	The constraint applies if the <i>layer1</i> shape touches a shape on the <i>tx_otherLayer</i> with the specified width range.

Example

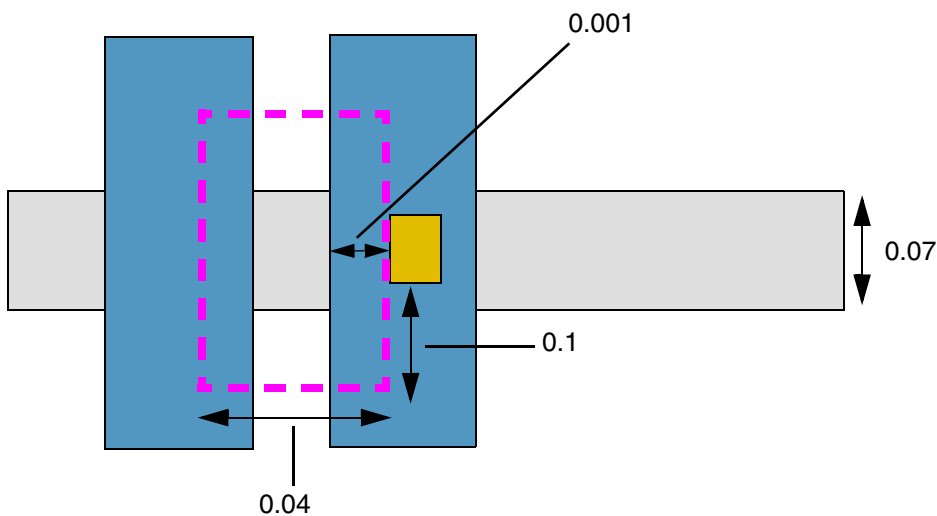
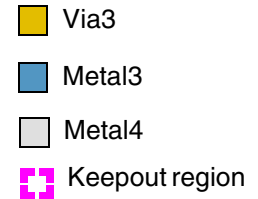
The constraint applies only if:

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

- the extension of Metal3 over Via3 edge is less than 0.002
- prl between Via3 and Metal3 is greater than -0.1
- the spacing between Via3 and Metal3 is less than or equal to 0.04
- the Metal4 shape, which intersects Via3, has width greater than 0.04

```
orderedSpacings (
  ( minCutEdgeSpacing "Via3" "Metal3"
    'prl -0.1
    'exceptEdgeExtension 0.002
    'horizontal
    'otherLayer "Metal4" 'otherLayerWidth ">0.04"
    0.04
  )
) ;orderedSpacings
```



The constraint applies because:

- the extension of Metal3 over the Via3 edge (0.001) is less than 0.002,
- prl between Via3 and Metal3 is greater than -0.1, and
- otherLayerWidth (for otherLayer Metal4, which intersects Via3) is greater than 0.04.

However, the constraint fails because the spacing between the Via3 shape and Metal3 is less than 0.04.

minCutRoutingSpacing (Two layers)

```
orderedSpacings (
  ( minCutRoutingSpacing tx_layer1 tx_layer2
    ['overlappingLayer layer1]
    ['horizontal | 'vertical]
    ['cutClass {f_width | (f_width f_length) | t_name}]
    ['exceptSameNet ['shortEdge ['shortEdgePr1 f_pr1]]
      |
      'toConcaveCorner ['pr1 f_pr1]
      | 1
      'toDiffConcaveCorner
    ]
    ['enclosingLayer tx_enclLayer
      ['viaEdgeType x_viaEdgeType]
      ['enclosingLayerWidth f_enclWidth
        ['anyOppositeExtension f_oppExtension]
      ]
    ]
    ['widthHorizontal | 'widthVertical]
    ['parallelEdgeWithin f_parWithin 'otherExtension f_otherExtension]
    ['parallelExtension f_parExtension]
    ['maskOverlap | 'metalOverlap]
    ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    ['deltaVoltage f_voltage]
    f_spacing
  )
) ;orderedSpacings
```

Specifies the minimum spacing between a shape on a cut layer (*layer1*) and a shape on a routing layer (*layer2*). This constraint can be used to define the spacing between specific cut classes and neighboring metal shapes.

The cut shape to concave corner spacing applies only for a wire with default wire width containing the cut if all of the following conditions are true:

- The cut has zero extension on two opposite sides and a *layer2* extension less than *otherExtension* on one of the other two opposite sides.
- The via cut has two different-metal shapes on two opposite sides—the sides with zero extension—with parallel run length greater than *pr1* with the via cut.
- The spacing from a cut edge with zero extension to the below-metal shape is equal to (*parWithin* - 0.001) and to the above-metal shape is less than *parWithin*.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Values

<code>tx_layer1</code>	The first layer (cut layer) on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<code>tx_layer2</code>	The second layer (routing layer) on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<code>f_spacing</code>	The minimum required Euclidean spacing between the two layers.

Parameters

<code>'overlappingLayer layer1</code>	The constraint applies only when this metal layer overlaps both via and metal.
<code>'horizontal 'vertical</code>	(Virtuoso Layout Suite EXL and higher tiers) The direction in which spacing is measured. If the direction is not specified, the spacing is measured in all applicable directions. Type: Boolean
<code>'cutClass {f_width (f_width f_length) t_name}</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <code>f_width</code>: Width ■ <code>f_length</code>: Length ■ <code>t_name</code>: Name of the cut class
<code>'exceptSameNet</code>	The constraint does not apply to shapes on the same net. Type: Boolean
<code>'shortEdge</code>	The constraint applies only between a metal edge and a short edge of a rectangular cut shape. Type: Boolean

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

`'shortEdgePr1 f_pr1`

(Virtuoso Advanced Node for Layout Only) The constraint applies between a metal edge and a short edge of a rectangular cut shape only if the parallel run length between them is greater than this value.

`'toConcaveCorner`

The constraint applies between a cut shape and a concave corner of the enclosing metal shape.

Type: Boolean

`'pr1 f_pr1`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the cut edges with zero extension have parallel run length greater than this value with the neighboring shapes on the layers above and below the cut layer.

The parallel run length is measured in an orthogonal direction to the spacing direction, if specified.

Type: Boolean

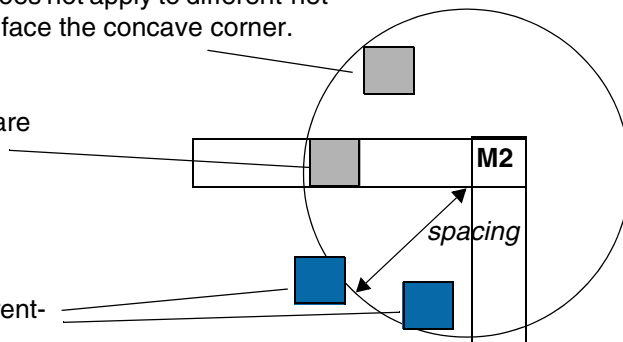
`'toDiffConcaveCorner`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only between a cut and a different-metal concave corner.

The constraint does not apply to different-net cuts that do not face the concave corner.

Same-net cuts are exempted.

The constraint applies to different-net cuts.



`'enclosingLayer tx_enclLayer`

(Virtuoso Advanced Node for Layout Standard) The enclosing layer for `'viaEdgeType` extension checks.

Type: String (layer name) or Integer (layer number)

`'viaEdgeType x_viaEdgeType`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to the cut edges of this type (the type is defined in the `viaEdgeType` constraint).

`'enclosingLayerWidth f_enclWidth`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the width of the enclosing shape on layer `enclLayer` is greater than or equal to this value.

`'anyOppositeExtension f_oppExtension`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if a cut shape has extensions less than this value on any two opposite sides.

`'widthHorizontal | 'widthVertical`

(Virtuoso Advanced Node for Layout Only) The direction in which the width of the wire specified by `enclosingLayer` parameter is measured.

`'parallelEdgeWithin f_parWithin`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if the neighboring shape on the metal layer below is at a distance equal to this value minus 0.001 (`parWithin - 0.001`) and the neighboring shape on the metal layer above is at a distance less than this value from the cut shape edges with zero extension.

`'otherExtension f_otherExtension`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if a cut shape with zero extension on two opposite sides has an extension less than this value on the other two opposite sides.

`'parallelExtension f_parExtension`

(Virtuoso Advanced Node for Layout Standard) The cut edges that do not fulfill the overhang requirement specified by `minExtensionEdge` are extended on both sides by this value before the spacing required by `minCutRoutingSpacing` is applied between an extended cut edge and a neighboring shape on `layer2`.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

'maskOverlap |
'metalOverlap

The constraint applies as follows:

- 'maskOverlap: (Virtuoso Advanced Node for Layout Standard) The constraint applies between a cut shape and the edge of an overlap between two metal shapes on two different masks.
- 'metalOverlap: (Virtuoso Layout Suite EXL and higher tiers) The constraint applies between a cut shape and the edge of the overlap of metal shapes.

Type: Boolean

'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- 'outsideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
| 'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)

The 'insideLayers and 'outsideLayers parameters are limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

`'deltaVoltage f_voltage`

(Virtuoso Advanced Node for Layout Only) The constraint applies if both the cut shape and the metal shape have voltages and the delta voltage between *layer1* (the cut shape) and *layer2* (the metal shape) is greater than this value.

The delta voltage is calculated as follows:

- For asynchronous nets:

$\max(V1_{\max}, V2_{\max}) - \min(V1_{\min}, V2_{\min})$

- For synchronous nets:

$\max(|V1_{\max} - V2_{\max}|, |V1_{\min} - V2_{\min}|)$

Examples

- [Example 1: minCutRoutingSpacing with cutClass, exceptSameNet and shortEdge](#)
- [Example 2: minCutRoutingSpacing with cutClass, toConcaveCorner, prl, parallelEdgeWithin, and otherExtension](#)
- [Example 3: minCutRoutingSpacing with enclosingLayer, viaEdgeType, and enclosingLayerWidth](#)
- [Example 4: minCutRoutingSpacing with cutClass and maskOverlap](#)
- [Example 5: minCutRoutingSpacing with anyOppositeExtension and deltaVoltage](#)
- [Example 6: minCutRoutingSpacing with metalOverlap](#)

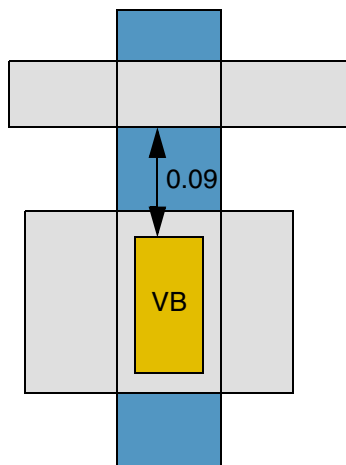
Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

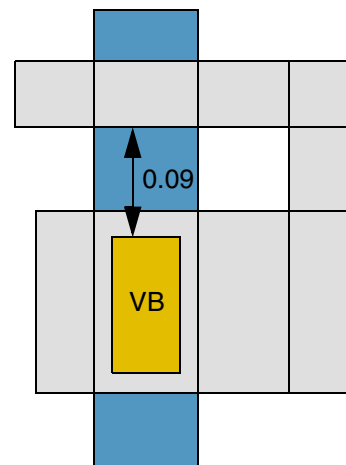
Example 1: *minCutRoutingSpacing* with *cutClass*, *exceptSameNet* and *shortEdge*

The spacing between the short edge of a Via1 via cut of type VB (0.4x0.6) and a Metal2 shape must be at least 0.1 if the via cut and the Metal2 wire are not on the same net.

```
orderedSpacings(  
  ( minCutRoutingSpacing "Via1" "Metal2"  
    'cutClass "VB"  
    'exceptSameNet 'shortEdge  
    0.1  
  )  
);orderedSpacings
```



a) FAIL. The spacing between the short edge of the via cut to an edge of a Metal2 wire is only 0.09 (<0.1).



b) The constraint does not apply because the via cut and the neighboring Metal2 wire are both on the same net.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

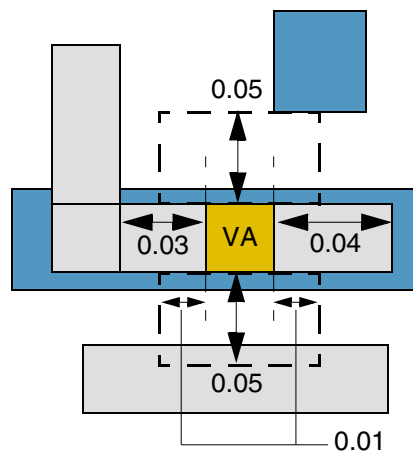
Example 2: *minCutRoutingSpacing* with *cutClass*, *toConcaveCorner*, *prl*, *parallelEdgeWithin*, and *otherExtension*

The spacing between a Via1 via cut of type VA and a concave corner of an enclosing Metal2 wire must be at least 0.04 if all of the following conditions are satisfied:

- The via cut has zero extension on two opposite sides and a Metal2 extension less than 0.06 on one of the two other opposite sides.
- The via cut has two different-metal wires on two opposite sides—the sides with zero extension—with parallel run length greater than -0.1 with the via cut.
- The spacing from a cut edge with zero extension to the below-metal (Metal1) wire is equal to 0.05 (0.051- 0.001) and to the above-metal (Metal2) wire is less than 0.051.

```
orderedSpacings (
  ( minCutRoutingSpacing "Via1" "Metal2"
    'cutClass "VA"
    'toConcaveCorner 'prl -0.01
    'parallelEdgeWithin 0.051 'otherExtension 0.06
    0.04
  )
) ;orderedSpacings
```

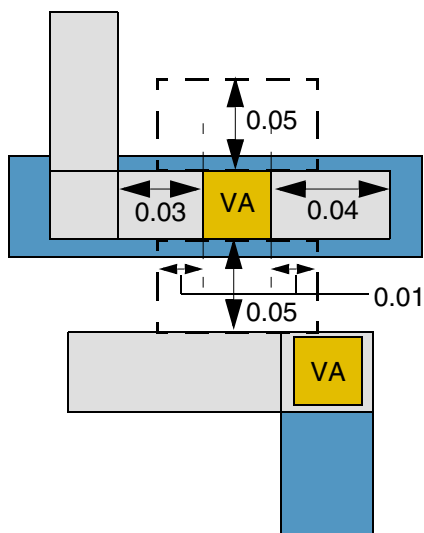
Metal2
 Via1
 Metal1



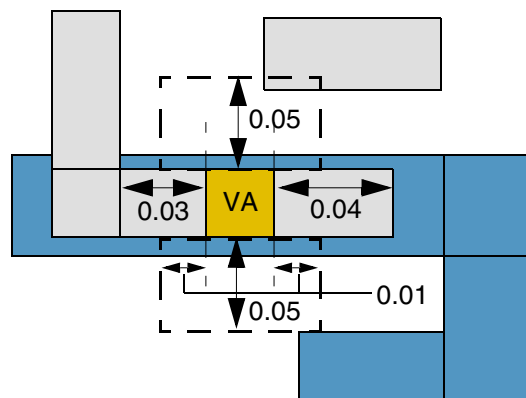
a) FAIL. The via cut is of type VA and has zero extensions on two opposite sides and extensions less than 0.06 on the other two opposite sides. The via cut also has two neighboring wires on opposite sides—the sides with zero extension—with parallel run length greater than -0.1 with the via cut. The distance of the below-metal (Metal1) wire is 0.05 (0.051-0.001) and the above-metal (Metal2) wire is less than 0.051 from the via cut. Therefore, via cut to concave corner spacing applies, but is only 0.03 (<0.04).

Virtuoso Technology Data Constraints Reference

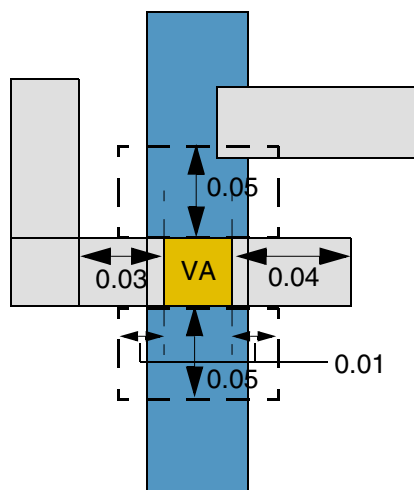
Spacing Constraints (Two Layers)



b) The constraint does not apply because the neighboring wires are not on opposite sides.



c) FAIL. The neighboring same-metal, same-net Metal1 wire is a valid neighbor. However, the spacing between the via cut and the concave corner is only 0.03 (< 0.04).



d) The constraint does not apply because the Metal1 wire is directly connected to the via cut.

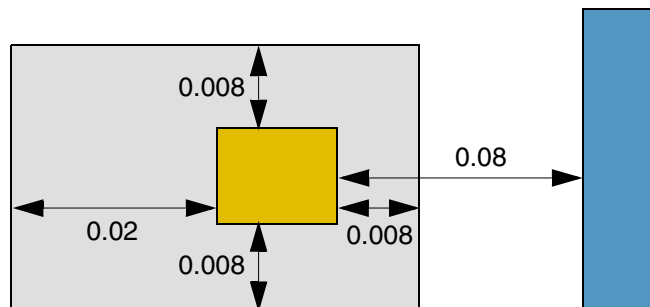
Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

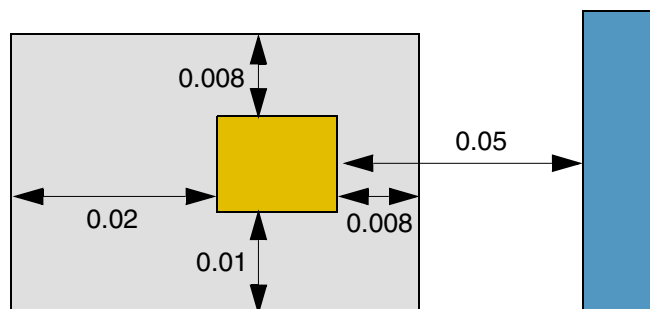
Example 3: *minCutRoutingSpacing* with *enclosingLayer*, *viaEdgeType*, and *enclosingLayerWidth*

The spacing between a Via1 via cut and a neighboring Metal1 wire must be at least 0.08. The constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.009.

```
spacings(  
  ( viaEdgeType "Via1"  
    'anyOppositeExtension 0.009  
    'negateAnyOppositeExtension  
      1  
    )  
  ) ;spacings  
orderedSpacings(  
  ( minCutRoutingSpacing "Via1" "Metal1"  
    'enclosingLayer "Metal2"  
    'viaEdgeType 1  
    'enclosingLayerWidth 0.0  
      0.08  
    )  
  ) ;orderedSpacings
```



a) The constraint does not apply because the via cut has two opposite edges with extensions less than or equal to 0.009.



b) FAIL. The top and right edges of the via cut have extensions 0.008 (less than or equal to 0.009), and are, therefore, of type viaEdgeType 1. However, the spacing between the right via cut edge and the neighboring Metal1 wire is only 0.05 (<0.08).

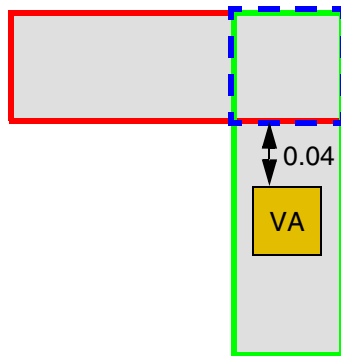
Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

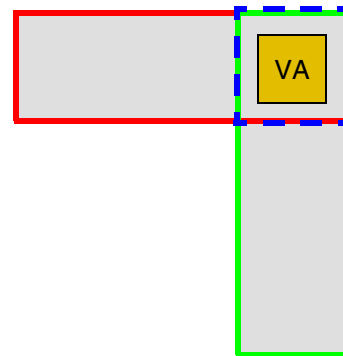
Example 4: *minCutRoutingSpacing* with *cutClass* and *maskOverlap*

The spacing between a Via1 via cut of type VA and the overlap of two Metal2 shapes on different masks must be at least 0.05.

```
orderedSpacings (  
  ( minCutRoutingSpacing "Via1" "Metal2"  
    'cutClass "VA"  
    'maskOverlap  
    0.05  
  )  
);orderedSpacings
```



a) FAIL. The spacing between the via cut and the overlap is only 0.04 (<0.05).



b) FAIL. The via cut cannot be inside the overlap.

Virtuoso Technology Data Constraints Reference

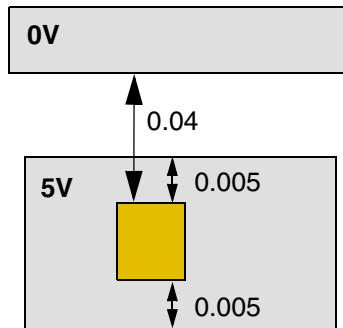
Spacing Constraints (Two Layers)

Example 5: *minCutRoutingSpacing* with *anyOppositeExtension* and *deltaVoltage*

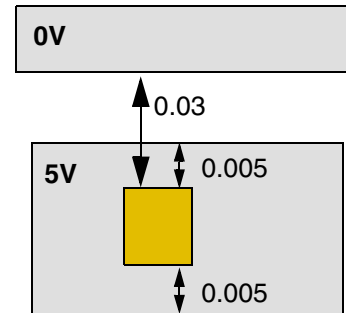
The spacing between a Via1 via cut and a neighboring Metal1 wire must be at least 0.04. The constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.0055. The distance between the via edge to another metal must be more than or equal to 0.04.

```
orderedSpacings (  
  ( minCutRoutingSpacing "Via1" "Metal2"  
    'anyOppositeExtension 0.0055  
    'enclosingLayer "Metal2"  
    'enclosingLayerWidth 0  
    'deltaVoltage 4.9  
    0.04
```

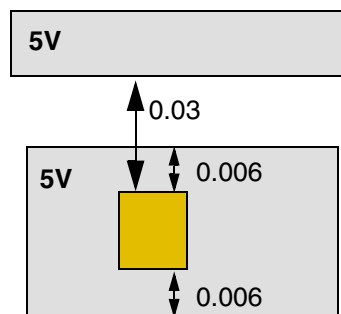
□ Metal2
■ Via1



a) PASS. The distance from the via edge to the metal edge is 0.04, which is greater than or equal to 0.04.



b) FAIL. The distance from the via edge to the metal is 0.03, which is less than 0.04.



c) PASS. The rule is not applicable because the delta voltage is 0.

Virtuoso Technology Data Constraints Reference

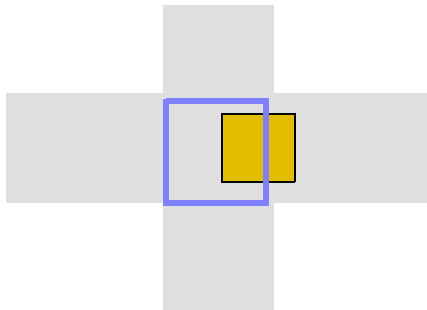
Spacing Constraints (Two Layers)

Example 6: minCutRoutingSpacing with metalOverlap

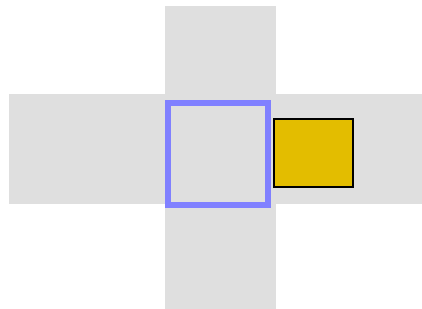
The `Via1` cut shape is not allowed to intersect the overlap area of the `Metal1` shapes.

```
orderedSpacings (  
  ( minCutRoutingSpacing "Via1" "Metal1"  
    'metalOverlap  
    0.0  
  )  
) ;orderedSpacings
```

□ Metal1
■ Via1



a) FAIL: The constraint applies and fails because the cut intersects the metal overlap area.



b) PASS: The constraint applies and the spacing between the cut and the metal overlap is met.

minInnerVertexSpacing (Two layers)

```
orderedSpacings (
    ( minInnerVertexSpacing tx_layer1 tx_layer2
      ['distance f_distance]
      f_spacing
    )
) ;orderedSpacings
```

Specifies the minimum spacing between the edges that form an inside corner of a *layer1* shape and the *layer2* edges that face the inside corner. At least one of the two *layer2* edges facing the inside corner must meet the minimum spacing requirement.

Optionally, this constraint does not apply if both *layer2* edges facing the inside corner are at a distance greater than or equal to *distance* from the *layer1* edges that form the inside corner.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum required spacing.

Parameters

'distance'	The constraint does not apply if both <i>layer2</i> edges adjacent to the <i>layer1</i> inside corner are at a distance greater than or equal to this value from the <i>layer1</i> edges that form the inside corner.

Virtuoso Technology Data Constraints Reference

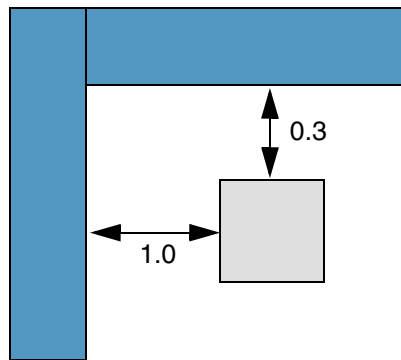
Spacing Constraints (Two Layers)

Example

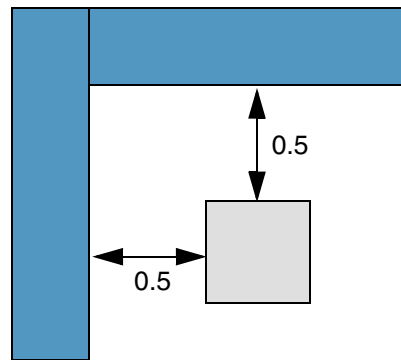
Metal1 and Metal2 shapes must satisfy a minimum spacing requirement of 0.3 and a minimum inner vertex spacing requirement of 1.0.

```
spacings (
  ( minSpacing "Metal1" "Metal2"
    0.3
  )
) ;spacings
orderedSpacings (
  ( minInnerVertexSpacing "Metal1" "Metal2"
    'distance 0.5
    1.0
  )
) ;orderedSpacings
```

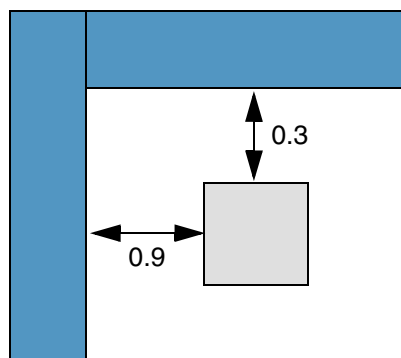
□ Metal2
■ Metal1



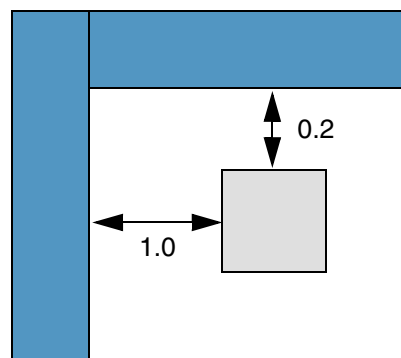
a) PASS. The top Metal2 edge satisfies the minimum spacing requirement of 0.3 and the left Metal2 edge satisfies the minimum inner vertex spacing requirement of 1.0.



b) PASS. The top and left Metal2 edges satisfy the minimum spacing requirement of 0.3. The `minInnerVertexSpacing` constraint does not apply because both edges are at a distance equal to 0.5 from the inside corner.



c) FAIL. The top Metal2 edge satisfies the minimum spacing requirement of 0.3, but the left Metal2 edge does not satisfy the minimum inner vertex spacing requirement of 1.0.

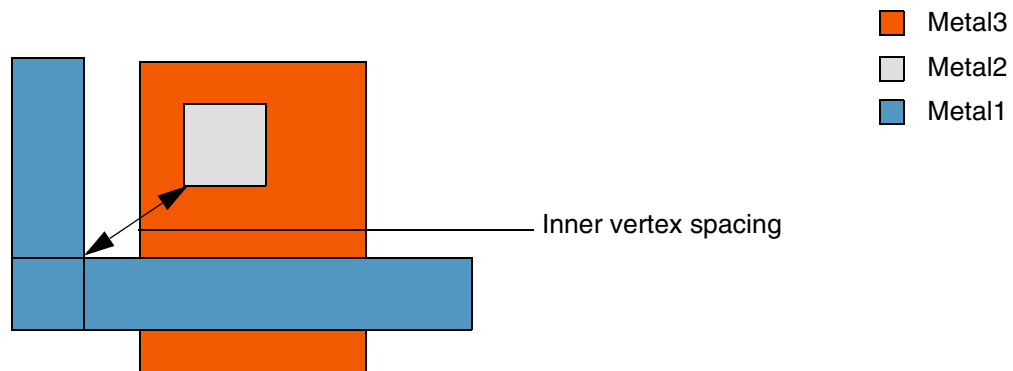


d) FAIL. The top Metal2 edge does not satisfy the minimum spacing requirement of 0.3.

minInnerVertexSpacing (Three layers)

```
orderedSpacings (
    ( minInnerVertexSpacing tx_layer1 tx_layer2 tx_layer3
      f_spacing
    )
) ;orderedSpacings
```

Specifies minimum spacing between a contact shape on *layer2* and an inner vertex of a *layer1* shape, provided both *layer1* and *layer2* shapes overlap the same *layer3* shape.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer3</i>	The third layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Parameters

None

Example

The spacing between an inner vertex on Poly and a contact on Cont must be at least 0.4 if both Poly and Cont shapes overlap the same Diff shape.

```
orderedSpacings (
  ( minInnerVertexSpacing "Poly" "Cont" "Diff"
    0.4
  )
) ;orderedSpacings
```

minNeighboringShapesSpacing

```
orderedSpacings (
  ( minNeighboringShapesSpacing tx_layer1 tx_layer2
    'widthRanges (g_widthRanges)
    'oppositeSpacingRanges (g_oppSpacingRanges)
    'lineEndSpacingRanges (g_lineEndSpacingRanges)
    ['otherLayer tx_otherLayer]
    f_spacing
  )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between a shape on *layer1* with width within the specified width ranges and a shape on *layer2* if there exist two neighboring *layer2* shapes:

- One with its short edge facing the *layer2* shape to which the spacing is being measured, at a distance in the range *oppSpacingRanges* from the *layer1* shape. The two *layer2* shapes must be on opposite sides of the *layer1* shape.
- The other facing an end-of-line edge of the *layer1* shape, at a distance in the range *lineEndSpacingRanges* from the *layer1* shape.

Optionally, the neighboring shapes can be on a third layer, *otherLayer*.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shapes must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Parameters

`'widthRanges (g_widthRanges)`

The constraint applies only if the width of the *layer1* shape falls in these width ranges.

Type: Floating-point values specifying width ranges

`'oppositeSpacingRanges (g_oppSpacingRanges)`

The constraint applies only if there exists a neighboring *layer2* shape, with its short edge facing the *layer2* shape to which the spacing is being measured, at a distance in this range from the *layer1* shape. The two *layer2* shapes must be on opposite sides of the *layer1* shape.

Type: Floating-point values specifying spacing ranges

`'lineEndSpacingRanges (g_lineEndSpacingRanges)`

The constraint applies only if there exists a *layer2* shape, facing an end-of-line edge of the *layer1* shape, at a distance in this range from the *layer1* shape.

Type: Floating-point values specifying spacing ranges

`'otherLayer tx_otherLayer`

The neighboring shapes must be present on this layer. Otherwise, neighboring shapes must be present on *layer2*.

Examples

- Example 1: minNeighboringShapesSpacing with widthRanges, oppositeSpacingRanges, and lineEndSpacingRanges
- Example 2: minNeighboringShapesSpacing with widthRanges, oppositeSpacingRanges, lineEndSpacingRanges, and otherLayer

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

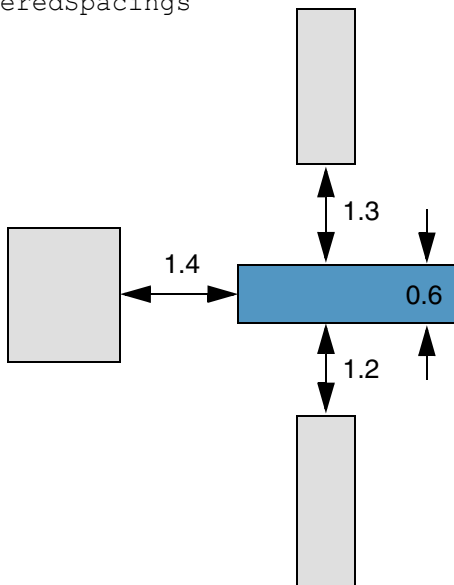
Example 1: *minNeighboringShapesSpacing* with *widthRanges*, *oppositeSpacingRanges*, and *lineEndSpacingRanges*

The spacing between a Metal1 shape and a Metal2 shape must be at least 1.2 if the following conditions are met:

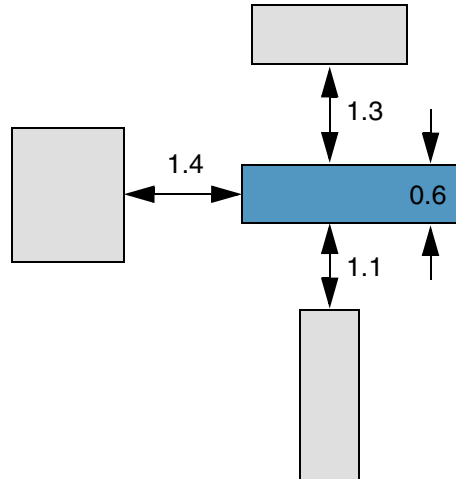
- There exists a neighboring Metal2 shape, with its short edge facing the Metal2 shape to which the spacing is being measured, at a distance greater than or equal to 1.0 and less than or equal to 1.4 from the Metal1 shape.
- There exists a Metal2 shape facing an end-of-line edge of the Metal1 shape at a distance greater than or equal to 1.3 and less than or equal to 1.5 from the Metal1 shape.

```
orderedSpacings (  
  ( minNeighboringShapesSpacing "Metal1" "Metal2"  
    'widthRanges ("[0.6 0.7]")  
    'oppositeSpacingRanges ("[1.0 1.4]")  
    'lineEndSpacingRanges ("[1.3 1.5]")  
    1.2  
  )  
) ;orderedSpacings
```

□ Metal2
■ Metal1



a) PASS. The constraint applies because the width of the Metal1 shape is in the range [0.6 0.7]; the short edge of the top Metal2 shape faces the Metal2 shape on the opposite side and its distance from the Metal1 shape is 1.3 (in the range [1.0 1.4]); and the distance of the Metal2 shape on the left from an end-of-line edge of the Metal1 shape is 1.4 (in the range [1.3 1.5]). The distance between the Metal1 shape and the bottom Metal2 shape is 1.2.



b) The constraint does not apply because the short edge of the top Metal2 neighboring shape does not face the Metal2 shape that is at a distance of 1.1 from the Metal1 shape.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

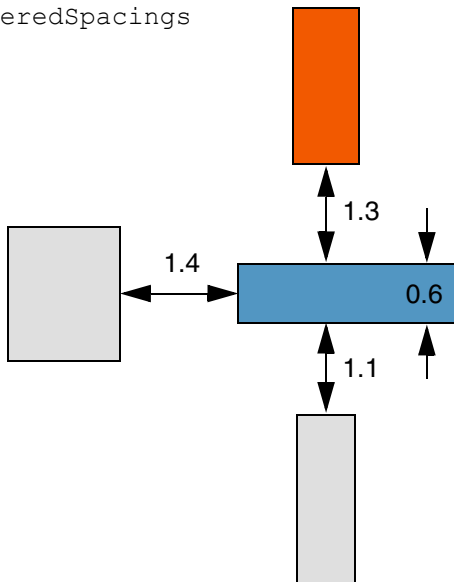
Example 2: *minNeighboringShapesSpacing* with *widthRanges*, *oppositeSpacingRanges*, *lineEndSpacingRanges*, and *otherLayer*

The spacing between a Metal1 shape and a Metal2 shape must be at least 1.2 if the following conditions are met:

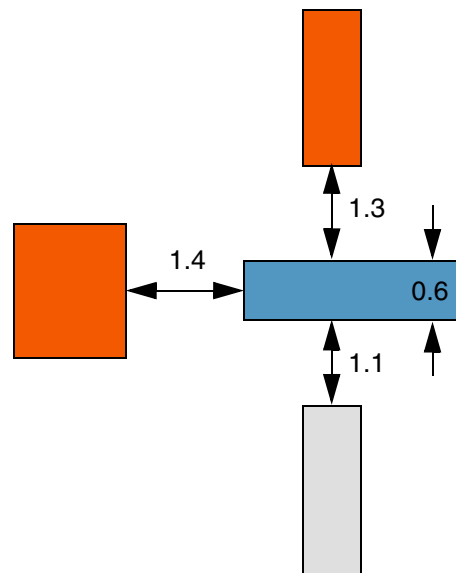
- There exists a neighboring Metal3 shape, with its short edge facing the Metal2 shape to which the spacing is being measured, at a distance greater than or equal to 1.0 and less than or equal to 1.4 from the Metal1 shape.
- There exists a Metal3 shape facing an end-of-line edge of the Metal1 shape at a distance greater than or equal to 1.3 and less than or equal to 1.5 from the Metal1 shape.

```
orderedSpacings (  
  ( minNeighboringShapesSpacing "Metal1" "Metal2"  
    'widthRanges ("[0.6 0.7]")  
    'oppositeSpacingRanges ("[1.0 1.4]")  
    'lineEndSpacingRanges ("[1.3 1.5]")  
    'otherLayer "Metal3"  
    1.2  
  )  
) ;orderedSpacings
```

■ Metal3
■ Metal2
■ Metal1



a) The constraint does not apply because the neighboring shape facing the end-of-line edge of the Metal1 shape is not on Metal3.



b) FAIL. The constraint applies because the width of the Metal1 shape is in the range [0.6 0.7]; the short edge of the top Metal3 shape faces the Metal2 shape on the opposite side and its distance from the Metal1 shape is 1.3 (in the range [1.0 1.4]); and the distance of the Metal2 shape on the left from an end-of-line edge of the Metal1 shape is 1.4 (in the range [1.3 1.5]). However, the distance between the Metal1 shape and the Metal2 shape is only 1.1 (<1.2).

minNeighborVoltageSpacing (Two layers)

```
spacings (
  ( minNeighborVoltageSpacing tx_layer1 tx_layer2
    ['horizontal | 'vertical]
    'otherLayer tx_otherLayer
    'otherDistance f_otherDistance
    'deltaVoltage f_voltage
    f_spacing
  )
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the spacing between two shapes, that meets the delta voltage condition, based on a neighboring shape.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shapes must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Parameters

`'horizontal | 'vertical`

The direction in which the spacing is measured. If the direction is not specified, spacing is measured in any direction.

`'otherLayer tx_otherLayer`

The layer that is used for `f_otherDistance`.

`'otherDistance f_otherDistance`

The constraint only applies when there are shapes on `tx_otherLayer` within this distance from the shape on `layer1` on the same side where the spacing is measured.

`'deltaVoltage f_voltage`

The constraint applies if the delta voltage between the `layer1` and `layer2` shapes is greater than this value.

The delta voltage is calculated as following:

- For asynchronous nets:

$\max(V1_{\max}, V2_{\max}) - \min(V1_{\min}, V2_{\min})$

- For synchronous nets:

$\max(|V1_{\max} - V2_{\max}|, |V1_{\min} - V2_{\min}|)$

Example

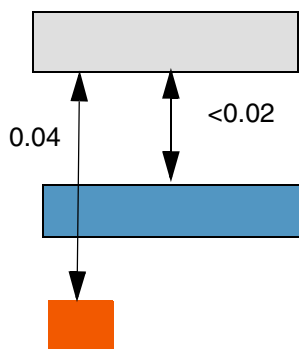
In this example, the maximum delta voltage is assumed to be > 1.1 . A minimum vertical spacing of 0.03 is required between shapes on layers `M1` and `M3` if there is an `M2` shape within 0.02 of the `M1` shape on the same side as the `M3` shape.

Virtuoso Technology Data Constraints Reference

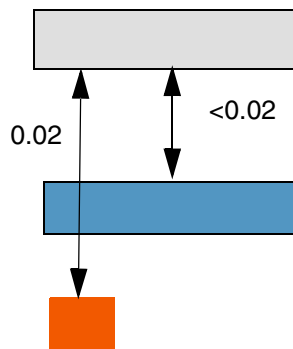
Spacing Constraints (Two Layers)

```
spacings (  
  ( minNeighborVoltageSpacing "M1" "M3"  
    'vertical  
    'otherLayer "M2"  
    'otherDistance 0.02  
    'deltaVoltage 1.1  
    0.03  
  )  
) ;spacings
```

□ M1
■ M"
■ M3



Pass: Spacing is 0.04 which is greater than the rule, ≥ 0.03



Fail: Spacing is 0.02 which is less than the rule, ≥ 0.03

minOppositeClearance (Two layers)

```
orderedSpacings (
    ( minOppositeClearance tx_cut1 tx_cut2
      'prl g_prl
      g_spacing
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Only) Specifies the cut spacing between a cut of any cut class on a layer and two cuts of any cut class on a second layer.

Values

<i>tx_cut1</i>	The first cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cut2</i>	The second cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_spacing</i>	The minimum spacing required between the via cuts.

Parameter

<i>'prl g_prl</i>	The constraint applies only if two cuts on <i>tx_cut2</i> individually have a parallel run length greater than or equal to this value to <i>tx_cut1</i> .

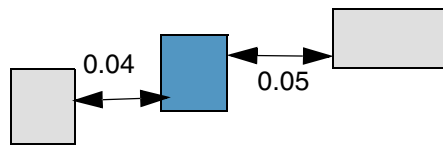
Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

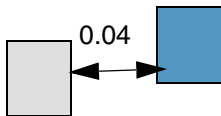
Example

```
orderedSpacings(  
  ( minOppositeClearance "Via2" "Via1"  
    'prl 0.01  
    0.05  
  )  
) ;orderedSpacings
```

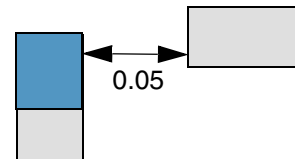
□ Via1
■ Via2



a) Violation. A Via2 cut has two Via1 cuts on opposite sides with $prl \geq 0.01$ on each of them, and the Via1 cut on the left has a spacing of 0.04 (< 0.05).



b) The constraint does not apply because there is only one Via1 cut.



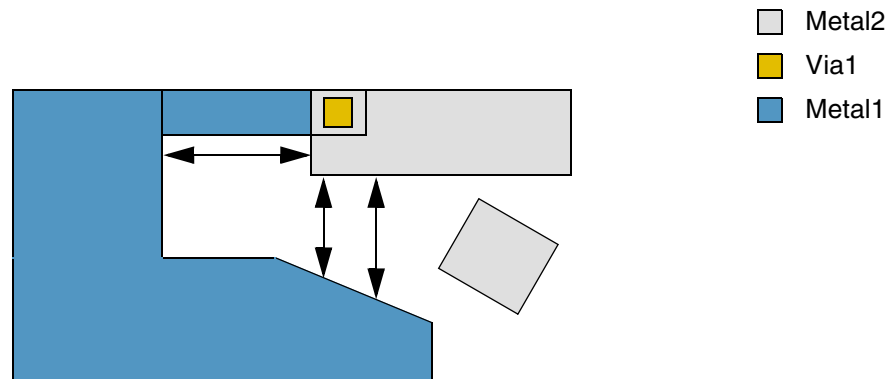
a) Violation. The overlapping same-net Via1 cut is a violation when another different-net Via2 cut fulfills the prl condition.

minSameNetSpacing (Two layers)

```
spacings (
    ( minSameNetSpacing tx_layer1 tx_layer2
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between shapes on the same net, but on different layers.

This constraint is required if the same-net spacing is less than the different-net spacing specified by the `minSpacing (Two layer)` constraint or if vias on different layers follow special stacking rules.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shapes must be greater than or equal to this value.

Parameters

None

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example

The same-net spacing between shapes on Metal1 and Metal2 must be at least 1.0.

```
spacings (
  ( minSameNetSpacing "Metal1" "Metal2"
    1.0
  )
) ;spacings
```


minSideSpacing (Two layers)

```
orderedSpacings (
    ( minSideSpacing tx_layer1 tx_layer2
      ['shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
        | 'longSideToLongSide | 'longSideToAnySide
      ]
      ['widthRanges (g_ranges)]
      ['otherWidthRanges (g_otherRanges)]
      ['horizontal | 'vertical]
      ['exceptOverlap f_exceptOverlap | 'exceptExactOverlap
f_exceptExactOverlap]
      ['cornerEuclidian]
      ['mask1 | 'mask2 | 'mask3]
      ['otherMask1 | 'otherMask2 | 'otherMask3]
      ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      ['deltaVoltage f_voltage]
      ['prl f_prl | 'prlRange g_prlRange]
      f_spacing
    )
) ;orderedSpacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the minimum spacing between two shapes on two different layers based on which side of the *layer1* shape faces which side of the *layer2* shape. Sides can be of type short, long, or any.

If the parallel run length between the shapes is less than zero, spacing can optionally be measured corner-to-corner using Euclidean measure type. By default, spacing is measured using Manhattan measure type.

Note: Sides of non-rectangular shapes and squares are of type "any" (that is, they are not considered as short or long).

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer name) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

f_spacing The spacing between the shapes must be greater than or equal to this value.

Parameters

'shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
| 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide

The sides between which spacing is measured.

- 'shortSideToShortSide: Spacing is measured between a short side of the *layer1* shape and a short side of the *layer2* shape.
- 'shortSideToLongSide: Spacing is measured between a short side of the *layer1* shape and a long side of the *layer2* shape.
- 'shortSideToAnySide: Spacing is measured between a short side of the *layer1* shape and any side of the *layer2* shape.
- 'longSideToLongSide: Spacing is measured between a long side of the *layer1* shape and a long side of the *layer2* shape.
- 'longSideToAnySide: Spacing is measured between a long side of the *layer1* shape and any side of the *layer2* shape.
- 'anySideToAnySide: Spacing is measured between any side of the *layer1* shape and any side of the *layer2* shape.

'widthRanges (*g_widthRanges*)

The constraint applies only if the width of the *layer1* shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

`'otherWidthRanges (g_otherWidthRanges)`

The constraint applies only if the width of the *layer2* shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint

`'horizontal | 'vertical`

The direction in which spacing is measured. If direction is not specified, spacing is measured in any direction.

Type: Boolean

`'exceptOverlap f_exceptOverlap | 'exceptExactOverlap
f_exceptExactOverlap`

- **exceptOverlap:** (Virtuoso Advanced Node for Layout Only) For positive *exceptOverlap* values, the constraint does not apply if the overlap between the shapes in the spacing direction is greater than or equal to the specified value. For negative *exceptOverlap* values, the constraint does not apply if the distance between the two shapes in the spacing direction is less than the absolute value of *exceptOverlap*.
- **exceptExactOverlap:** (Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply if the overlap between the shapes in the spacing direction is exactly equal to the specified value. Only positive values are allowed.

`'cornerEuclidian`

(Virtuoso Advanced Node for Layout Only) The corner-to-corner spacing (when parallel run length is less than zero) between shapes is measured using Euclidean measure type (the spacing halo has rounded corners).

The default measure type is Manhattan.

Type: Boolean

`'mask1 | 'mask2 | 'mask3`

(Virtuoso Advanced Node for Layout Only) The constraint applies to this mask on *layer1*.

`'otherMask1 | 'otherMask2 | 'otherMask3`

(Virtuoso Advanced Node for Layout Only) The constraint applies to this mask on *layer2*.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

```
'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- 'outsideLayers: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'deltaVoltage f_voltage
```

(Virtuoso Advanced Node for Layout Only) The constraint applies if both the cut shape and the metal shape have voltages and the delta voltage between the *layer1* shape and the *layer2* shape is greater than this value.

The delta voltage is calculated as follows:

- For asynchronous nets:
$$\max(V1_{\max}, V2_{\max}) - \min(V1_{\min}, V2_{\min})$$
- For synchronous nets:
$$\max(|V1_{\max} - V2_{\max}|, |V1_{\min} - V2_{\min}|)$$

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

`'prl f_prl` The constraint applies only if the parallel run length between the two sides is greater than this value.

`'prlRange g_prlRange`
(Virtuoso Advanced Node for Layout Only) The constraint applies only if the parallel run length between the two sides falls in this range.

Type: Floating-point values specifying a range of parallel run lengths that trigger the constraint

Examples

- Example 1: minSideSpacing with shortSideToShortSide, prl, widthRanges, and otherWidthRanges
- Example 2a: minSideSpacing with exceptOverlap
- Example 2b: minSideSpacing with exceptOverlap
- Example 3: minSideSpacing with deltaVoltage and prlRange

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

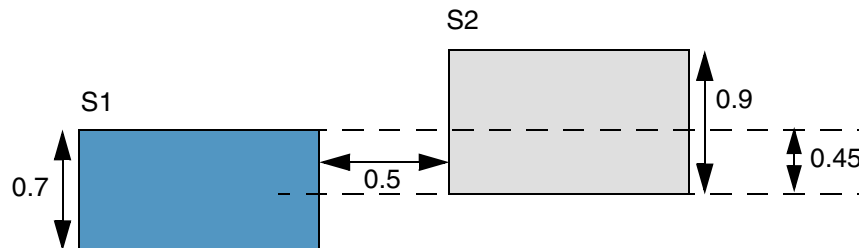
Example 1: *minSideSpacing* with *shortSideToShortSide*, *prl*, *widthRanges*, and *otherWidthRanges*

The spacing between the short sides of shapes S1 and S2 must be greater than or equal to 0.4 if the following conditions are met:

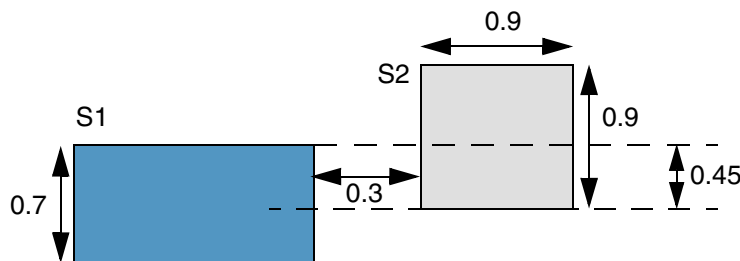
- The width of S1 is greater than or equal to 0.6 and less than or equal to 0.8.
- The width of S2 is greater than or equal to 0.9 and less than or equal to 1.0.
- The parallel run length between the sides is greater than 0.

```
orderedSpacings (
  ( minSideSpacing "Metal1" "Metal2"
    'shortSideToShortSide
    'prl 0.0
    'widthRanges ("[0.6 0.8]")
    'otherWidthRanges ("[0.9 1.0]")
    0.4
  )
) ;orderedSpacings
```

Metal2
 Metal1



a) PASS. The widths of both shapes fall in the specified width ranges. The parallel run length between the short sides is greater than zero and the spacing between the short sides is 0.5 (>0.4).



b) The constraint does not apply because the other shape (S2) with both sides equal to 0.9 is a square (its sides are of type any).

Virtuoso Technology Data Constraints Reference

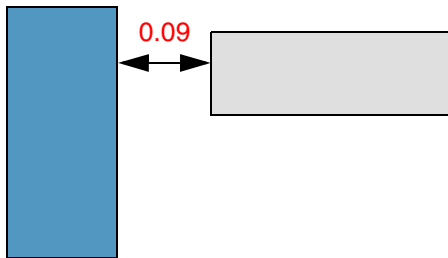
Spacing Constraints (Two Layers)

Example 2a: *minSideSpacing with exceptOverlap*

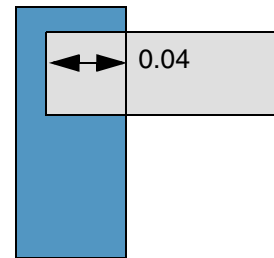
The spacing between Trim1 and Metal2 shapes must be at least 0.1, except when the horizontal overlap between the two shapes is greater than or equal to 0.05.

```
orderedSpacings (  
  ( minSideSpacing "Trim1" "Metal2"  
    'horizontal  
    'exceptOverlap 0.05  
    0.1  
  )  
) ;orderedSpacings
```

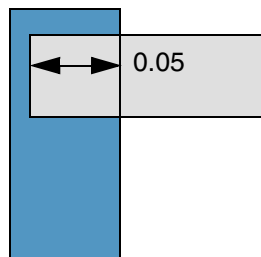
□ Metal2
■ Trim1



a) FAIL. The spacing between the two shapes is only 0.09 (<0.1).



b) FAIL. The overlap in the horizontal direction is less than 0.05. Therefore, the spacing requirement of 0.1 must be satisfied.



c) The constraint does not apply because the overlap between the two shapes in the horizontal direction is equal to 0.05.

Virtuoso Technology Data Constraints Reference

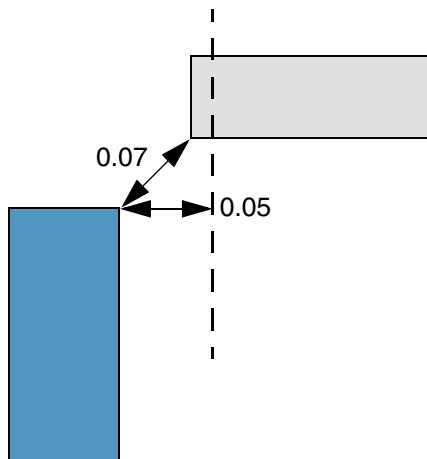
Spacing Constraints (Two Layers)

Example 2b: *minSideSpacing with exceptOverlap*

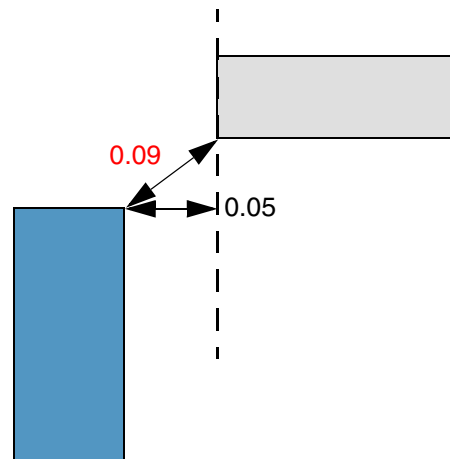
The spacing between Trim1 and Metal2 shapes must be at least 0.1, except when the horizontal distance between the two shapes is less than 0.05.

```
orderedSpacings (  
  ( minSideSpacing "Trim1" "Metal2"  
    'horizontal  
    'exceptOverlap -0.05  
    0.1  
  )  
) ;orderedSpacings
```

□ Metal2
■ Trim1



a) The constraint does not apply because the distance between the two shapes in the horizontal direction is less than 0.05.



b) FAIL. The distance between the two shapes in the horizontal direction is equal to 0.05. Therefore, the spacing requirement of 0.1 must be satisfied.

Virtuoso Technology Data Constraints Reference

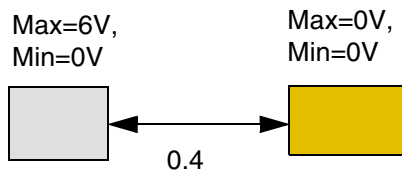
Spacing Constraints (Two Layers)

Example 3: *minSideSpacing* with *deltaVoltage* and *prlRange*

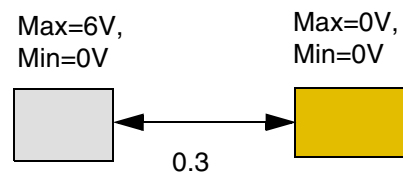
The spacing between two vias can be specified separately for different parallel run lengths and voltage differences as illustrated in this example.

```
orderedSpacings (
  ( minSideSpacing "Via1" "Via2" 'deltaVoltage 4
    'horizontal 'prlRange ">0" 0.4 )
  ( minSideSpacing "Via1" "Via2" 'deltaVoltage 4
    'horizontal 'prlRange "[-.2 0)" 0.2 )
) ;orderedSpacings
```

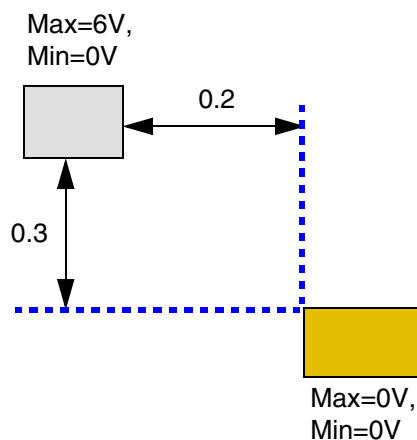




a) PASS. The constraints apply because *deltaVoltage* is 6V (which is more than 4V).



b) FAIL. The first constraint applies but fails because *prl* is more than 0 and spacing is 0.3 (which is less than 0.4).



c) PASS. The second constraint applies and passes because *prl* is less than 0 and spacing is 0.3 (which is more than 0.2).

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

minSpacing (Two layers)

```
spacings (
  ( minSpacing tx_layer1 tx_layer2
    ['orthogonalSpacing f_orthoSpacing]
    ['horizontal | 'vertical]
    ['overlapNotAllowed]
    ['crossingAllowed]
    ['cornerAllowed]
    ['length f_length]
    ['exceptSameMetal]
    ['insideLayers (tx_layer4 tx_layer5 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    ['outsideLayers (tx_layer4 tx_layer5 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    f_spacing ['manhattan] ['coincidentAllowed]
  )
) ;spacings

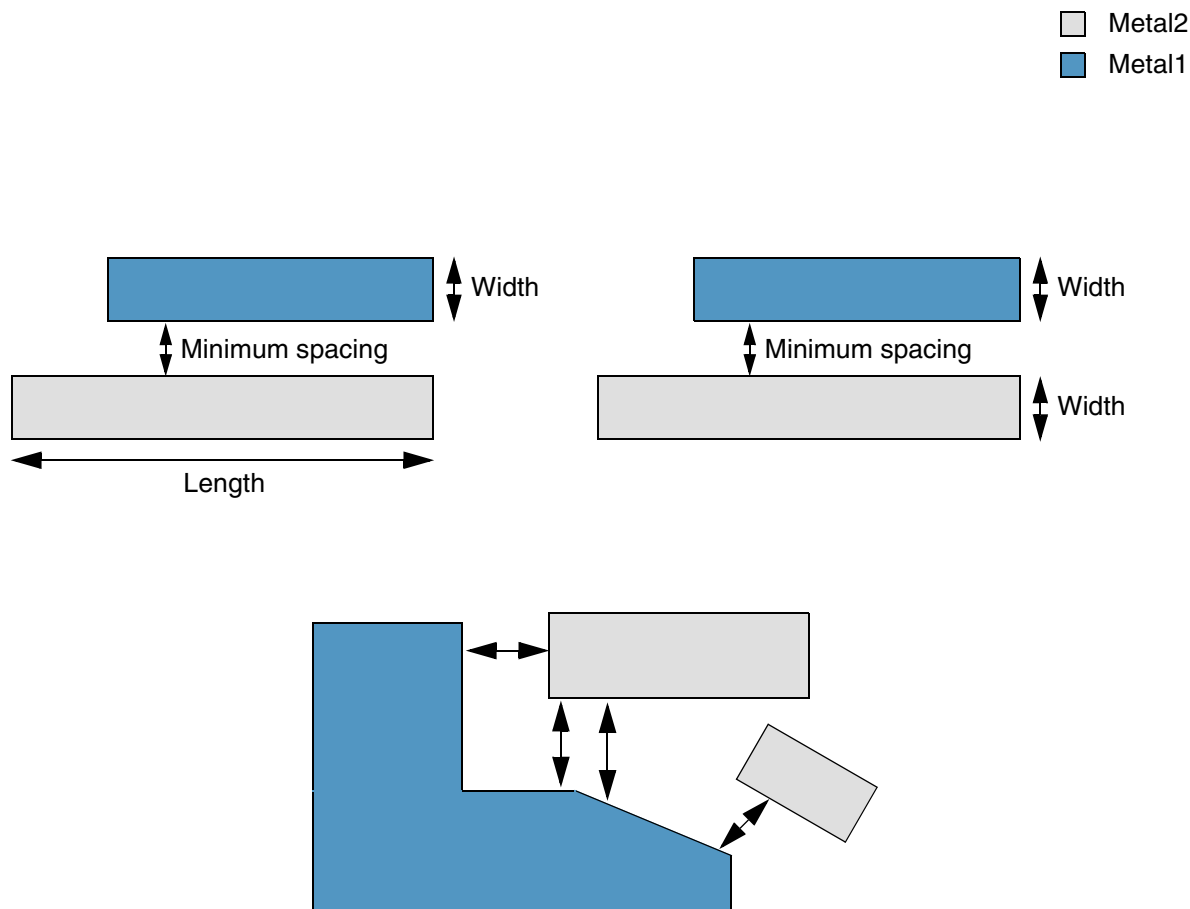
spacingTables (
  ( minSpacing tx_layer1 tx_layer2
    (( "width" nil nil "width" nil nil )
    ['horizontal | 'vertical]
    ['overlapNotAllowed]
    ['crossingAllowed]
    [f_default]
    )
    (g_table) ['manhattan]
  )
  ( minSpacing tx_layer1 tx_layer2
    (( "width" nil nil "length" nil nil )
    ['horizontal | 'vertical]
    ['overlapNotAllowed]
    ['crossingAllowed]
    [f_default]
    )
    (g_table) ['manhattan]
  )
  ( minSpacing tx_layer1 tx_layer2
    (( "twoWidths" nil nil "length" nil nil )
    ['horizontal | 'vertical]
    ['overlapNotAllowed]
    ['crossingAllowed]
    [f_default]
    )
    (g_table) ['manhattan]
  )
) ;spacingTables
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Specifies the minimum edge-to-edge spacing required between two adjacent, non-intersecting shapes on two different layers. The required spacing can be dependent on the direction in which the spacing is measured, on the width of both the shapes or on the width of the shapes and the parallel run length between them.

The figures below illustrate how the constraint is applied:

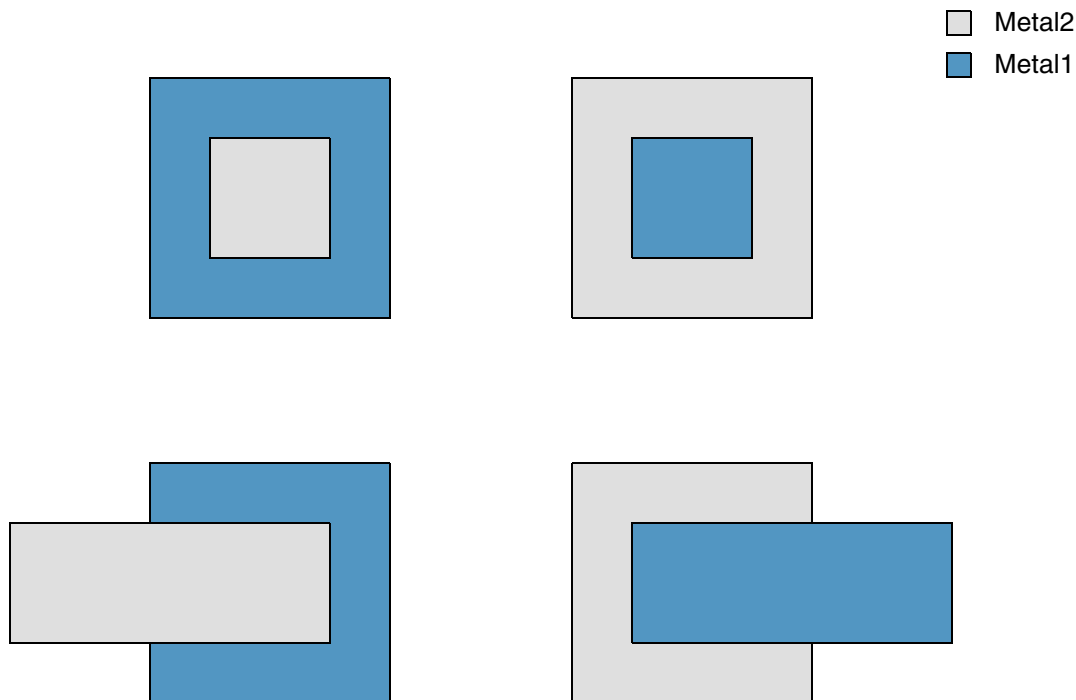


All distances indicated by arrows must be greater than or equal to the specified minimum spacing value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

The constraint does not apply in the scenarios illustrated by the figures below:



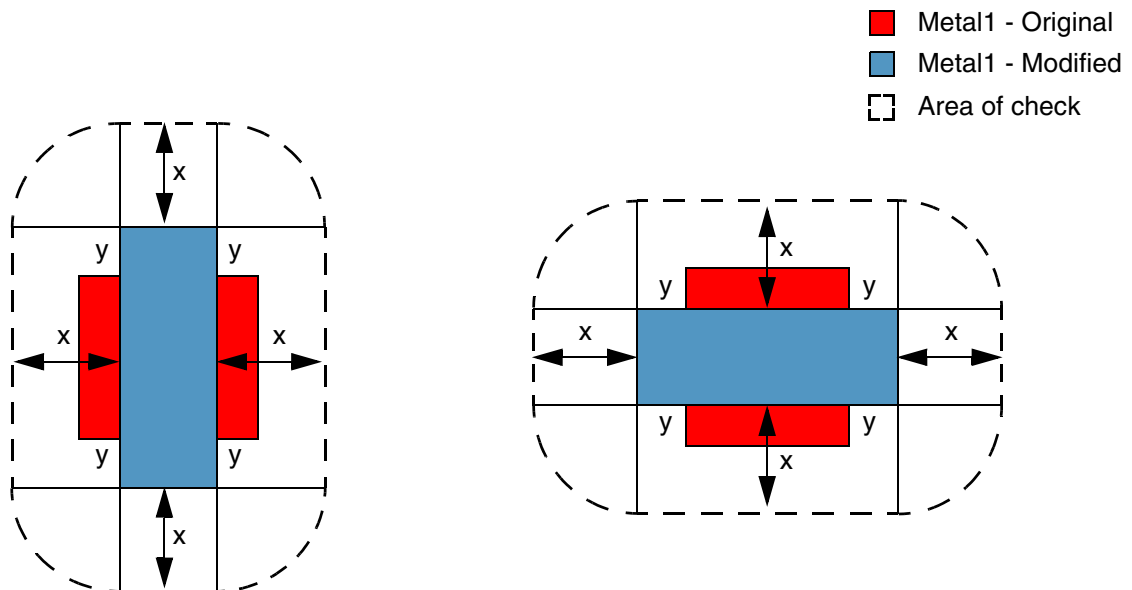
For some processes, the minimum spacing between a cut shape on one layer and a metal shape on a second layer on a different net is measured using a more complex method called orthogonal spacing. If the shapes are on the same net and orthogonal spacing is specified, the constraint does not apply.

The minimum orthogonal spacing check is determined by the following method: A pair of opposite sides of the *layer1* shape is grown by a distance equal to $\text{abs}(\text{minimum spacing} - \text{orthogonal spacing})/2$, denoted by "y" in the figure below. Similarly, the other pair of opposite sides of the *layer1* shape is shrunk by the same distance. The minimum required spacing between the modified *layer1* shape and the *layer2* shape is given by $(\text{minimum spacing} + \text{orthogonal spacing})/2$, denoted by "x" in the figure below.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

The spacing check is then repeated by shrinking the edges that were previously grown and growing the edges that were previously shrunk. The constraint is violated only if the *layer2* shape is found in the gray region of check shown in the figure below.



$$x = (\text{minimum spacing} + \text{orthogonal spacing})/2$$

$$y = \text{abs}(\text{minimum spacing} - \text{orthogonal spacing})/2$$

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing required between two shapes.
"width" nil nil "width" nil nil	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

"width" nil nil "length" nil nil

This identifies the index for *table*.

"twoWidths" nil nil "length" nil nil

This identifies the index for *table*.

g_table

The format of a 2-D table is as follows:

```
( (f_index1 f_index2) f_value
  ...
)
```

where,

- *f_index1* and *f_index2* both represent widths (of the two shapes), or *f_index1* is the width of the wider of the two shapes and *f_index2* is the parallel run length between the two shapes.
- *f_value* is the minimum spacing that applies when the width and parallel run length are both greater than or equal to the corresponding indexes.

Note: If *twoWidths* is specified, the lookup table establishes the minimum spacing between a pair of shapes based on the widths of both shapes and the parallel run length between them. Additionally, the widths and the parallel run length must be greater than the corresponding indexes (and not greater than or equal to).

Note: In a spacing table, all values should be explicitly specified. When a value is not specified, the default value *f_default* is used.

Type: A 2-D table specifying floating-point width, parallel run length, and spacing values

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Parameters

`'orthogonalSpacing f_orthoSpacing`

The different-net clearance between objects on a cut layer and objects on a routing layer must be equal to this value.

`'horizontal | 'vertical`

The direction in which the constraint applies. If direction is not specified, the constraint applies in any direction.

- `'horizontal`: The constraint applies only in the horizontal direction.

- `'vertical`: The constraint applies only in the vertical direction.

Type: Boolean

`'overlapNotAllowed`

(Virtuoso Advanced Node for Layout Standard) Shapes are not allowed to overlap. Otherwise, same-net shapes can overlap.

`'crossingAllowed`

(Virtuoso Advanced Node for Layout Only) The constraint does not apply to crossing shapes.

`'cornerAllowed`

(Virtuoso Advanced Node for Layout Only) The constraint does not apply if the corners of the two shapes touch.

`'length f_length`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to shapes with length less than this value.

`'exceptSameMetal`

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply to contiguous same-metal shapes.

Type: Boolean

`'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)`

`| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)`

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

(Virtuoso Layout Suite EXL and higher tiers) Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- `'outsideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN) |  
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Layout Suite EXL and higher tiers) Limits the `'insideLayers` and `'outsideLayers` parameters to the specified purposes.

Type: String (purpose name)

`'manhattan`

The constraint uses Manhattan distance, which allows a larger spacing at the corners.

By default, the constraint uses Euclidean measurement.

Type: Boolean

`f_default`

The spacing value to be used when no table entry applies or is specified (using *g_table*). Ensure that the default spacing value is sufficiently large to represent all missing spacing values in the table. Ideally, it should be greater than or equal to the largest specified spacing in the table.

`'coincidentAllowed`

The edges the two shapes can coincide. Otherwise, the shapes must meet the spacing requirement.

Type: Boolean

Examples

- [Example 1: minSpacing with vertical](#)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

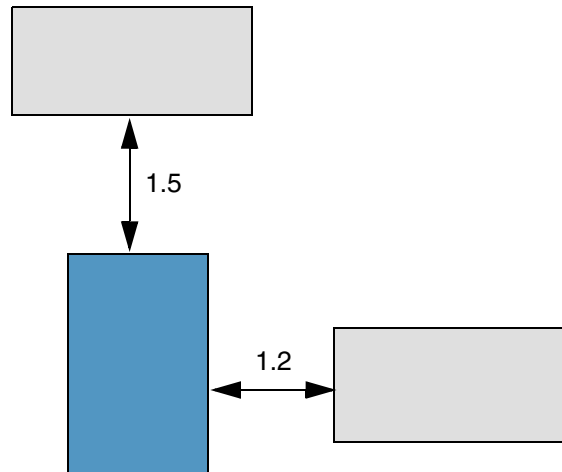
- [Example 2: minSpacing with orthogonalSpacing](#)
- [Example 3: minSpacing with crossingAllowed](#)

Example 1: minSpacing with vertical

The vertical spacing between Metal1 and Metal2 shapes must be at least 1.5.

```
spacings(  
  ( minSpacing "Metal1" "Metal2"  
    'vertical  
    1.5  
  )  
) ;spacings
```

□ Metal2
■ Metal1



PASS. The spacing between the Metal1 and Metal2 shapes in the vertical direction is 1.5.

Virtuoso Technology Data Constraints Reference

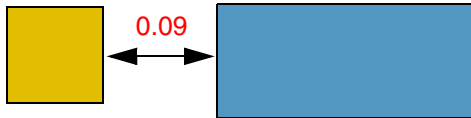
Spacing Constraints (Two Layers)

Example 2: minSpacing with orthogonalSpacing

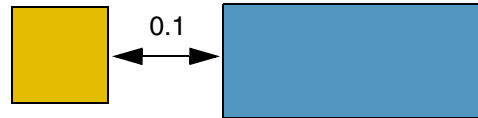
The spacing between Via1 and Metal1 shapes must be at least 0.1. Orthogonal spacing specified is 0.2, which means that Via1 shapes are grown and shrunk by 0.05 ($(0.2 - 0.1)/2 = 0.05$) and the spacing required in this case is 0.15 ($(0.2 + 0.1)/2 = 0.15$).

```
spacings (
  ( minSpacing "Via1" "Metal1"
    'orthogonalSpacing 0.2
    0.1
  )
) ;spacings
```

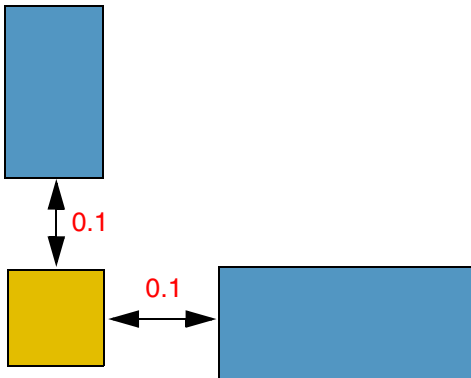
■ Via1
■ Metal1



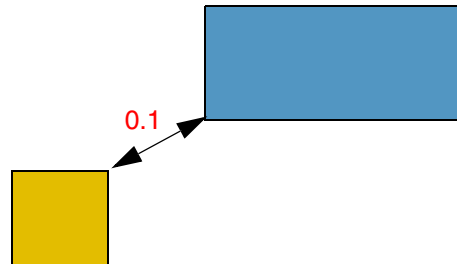
a) FAIL. The spacings between the two shapes after shrinking and growing the Via1 shape are 0.14 and 0.04, respectively, which are both less than the required spacing of 0.15.



b) PASS. The shrink operation gives a spacing of 0.15. Therefore, orthogonal spacing does not apply. The shapes satisfy the spacing requirement of 0.1.



c) FAIL. With both sets of shrink and grow operations, one Metal1 shape is found inside the area of check. Therefore, spacing of 0.15 applies in both directions and is not satisfied.



d) FAIL. The Metal1 shape is inside both checking regions.

Virtuoso Technology Data Constraints Reference

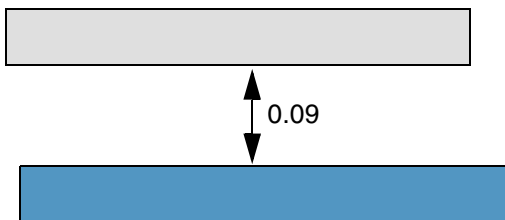
Spacing Constraints (Two Layers)

Example 3: *minSpacing* with *crossingAllowed*

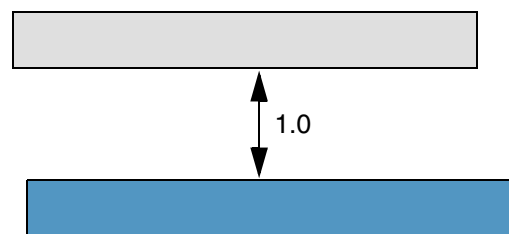
The spacing between Metal1 and Metal2 shapes must be at least 0.1. The constraint does not apply to crossing wires.

```
spacings(  
  ( minSpacing "Metal1" "Metal2"  
    'crossingAllowed  
    0.1  
  )  
) ;spacings
```

□ Metal2
■ Metal1



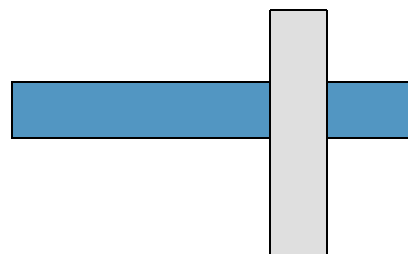
a) FAIL. The spacing between the two shapes is only 0.09 (<0.1).



b) PASS. The spacing between the two shapes is 1.0, which satisfies the constraint.



c) FAIL. Parallel shapes cannot touch.



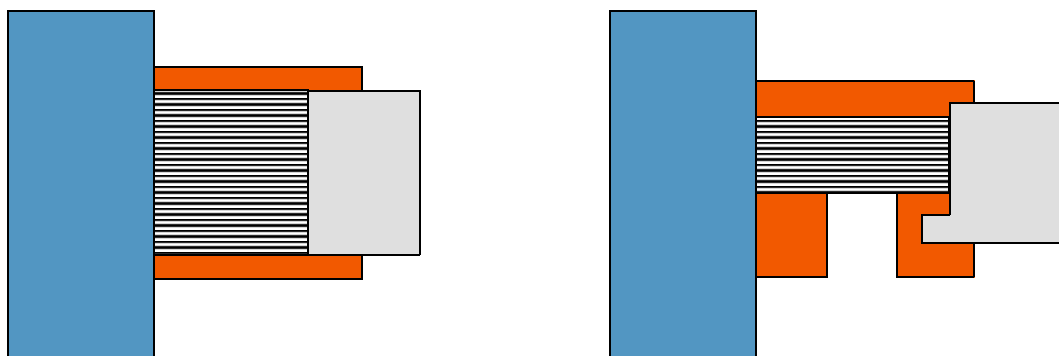
d) The constraint does not apply because 'crossingAllowed' is specified.

minSpacingOver

```
orderedSpacings (
  ( minSpacingOver tx_layer1 tx_layer2 tx_layer3
    ['insideLayers (tx_layer4 tx_layer5 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    'outsideLayers (tx_layer4 tx_layer5 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ]
  ['mask1 | 'mask2 | 'mask3]
  f_spacing
)
) ;orderedSpacings
```

Sets the minimum spacing between shapes on *layer1* and *layer2* in the region that is completely filled by a shape on *layer3*.

- Metal3
- Metal2
- Metal1
- Area to which the constraint applies



Values

tx_layer1

The first layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer3</i>	The reference layer. The placement of a <i>layer3</i> shape between <i>layer1</i> and <i>layer2</i> shapes determines how the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the shapes must be greater than or equal to this value.

Parameters

```
'insideLayers (tx_layer 4 tx_layer5 ... tx_layerN)
| 'outsideLayers (tx_layer4 tx_layer5 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- **'insideLayers:** The constraint applies when the region between the shapes on *layer1* and *layer2*, which is completely filled by a shape on *layer3*, overlaps a shape on one of these layers (*tx_layer4 tx_layer5 ... tx_layerN*).
- **'outsideLayers:** The constraint applies when the region between the shapes on *layer1* and *layer2*, which is completely filled by a shape on *layer3*, overlaps the region outside the shapes on one of these layers (*tx_layer4 tx_layer5 ... tx_layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The **'insideLayers** parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'mask1 | 'mask2 | 'mask3`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the shape on the third layer is of the specified color.

Type: Boolean

Example

Wherever a Poly shape completely fills the space between Metal1 and Metal2 shapes, the minimum spacing between Metal1 and Metal2 shapes must be at least 2.0.

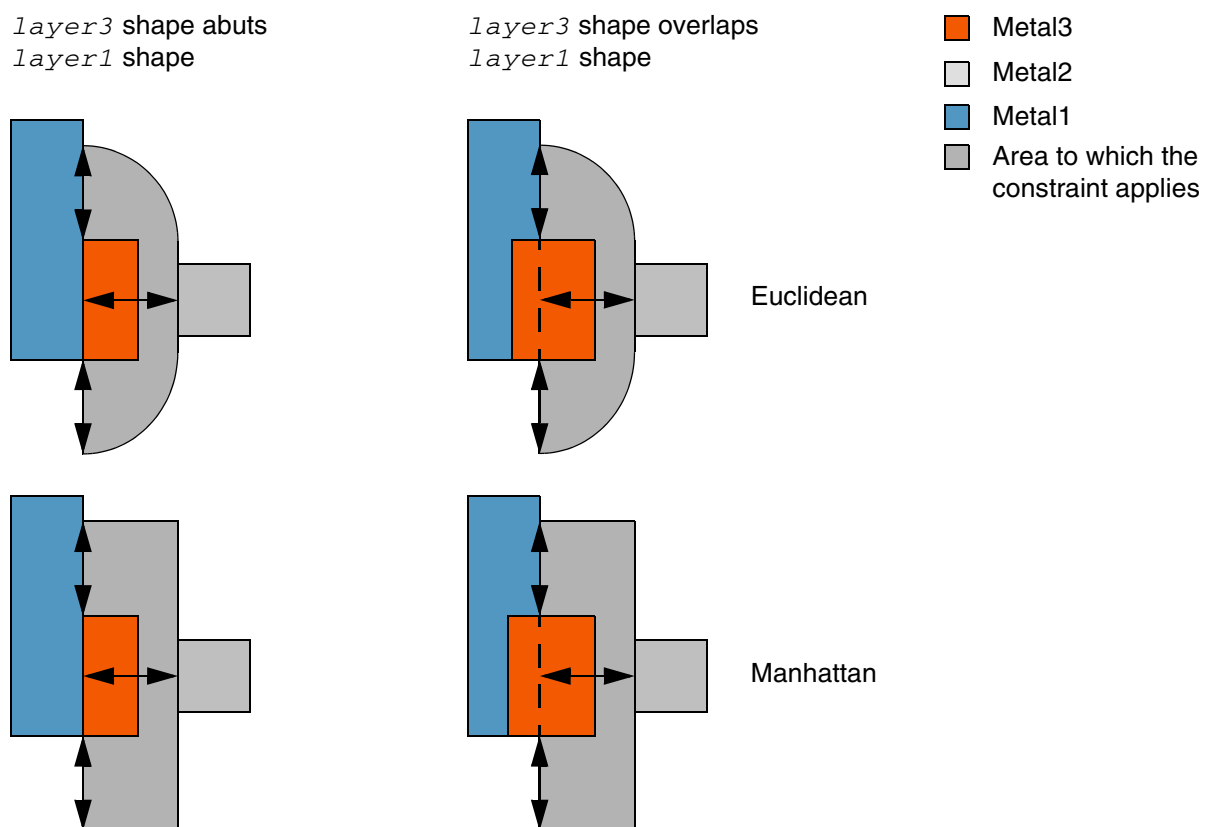
```
orderedSpacings(  
    ( minSpacingOver "Metal1" "Metal2" "Poly"  
      2.0  
    )  
) ;orderedSpacings
```

minTouchingDirSpacing

```
orderedSpacings (
  ( minTouchingDirSpacing tx_layer1 tx_layer2 tx_layer3
    f_spacing ['manhattan']
  )
  ( minTouchingDirSpacing tx_layer1 tx_layer2 tx_layer3
    (f_spacing f_oppSpacing) ['manhattan']
  )
) ;orderedSpacings
```

Specifies the minimum spacing between *layer1* and *layer2* shapes in the direction perpendicular to the *layer1* edge that is abutted or overlapped by a shape on *layer3*.

When only *spacing* is specified, the area to which the constraint applies is determined as shown in the following figure.

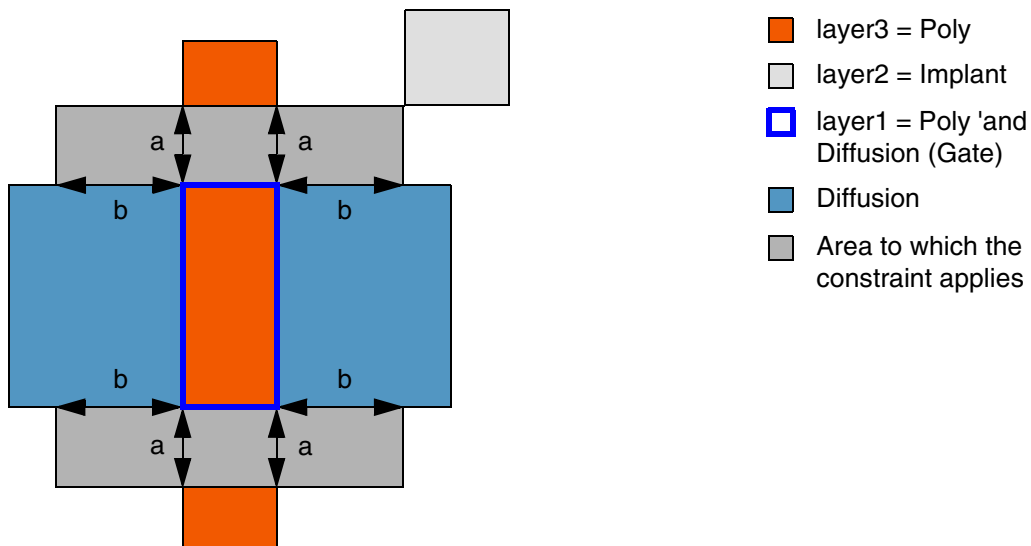


If both *spacing* and *oppSpacing* are specified, *spacing* describes the minimum spacing in the direction perpendicular to the *layer1* edge abutted or overlapped by the *layer3* shape and *oppSpacing* describes the minimum spacing parallel to this *layer1* edge. The distance must be measured as Manhattan, as shown below.

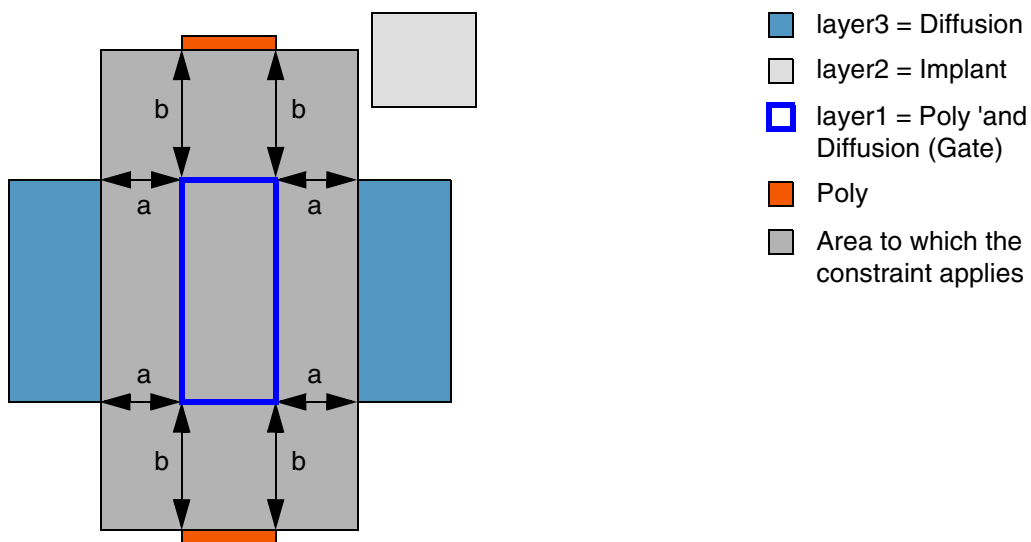
Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

In this first example, in which *layer3* is set to Poly, the spacing to the Implant layer must be at least *a* (*spacing*) in the direction of the touching Poly layer (that is, in the direction perpendicular to the *layer1* edge overlapped by the *layer3* shape) and *b* (*oppSpacing*) in the direction parallel to this *layer1* edge.



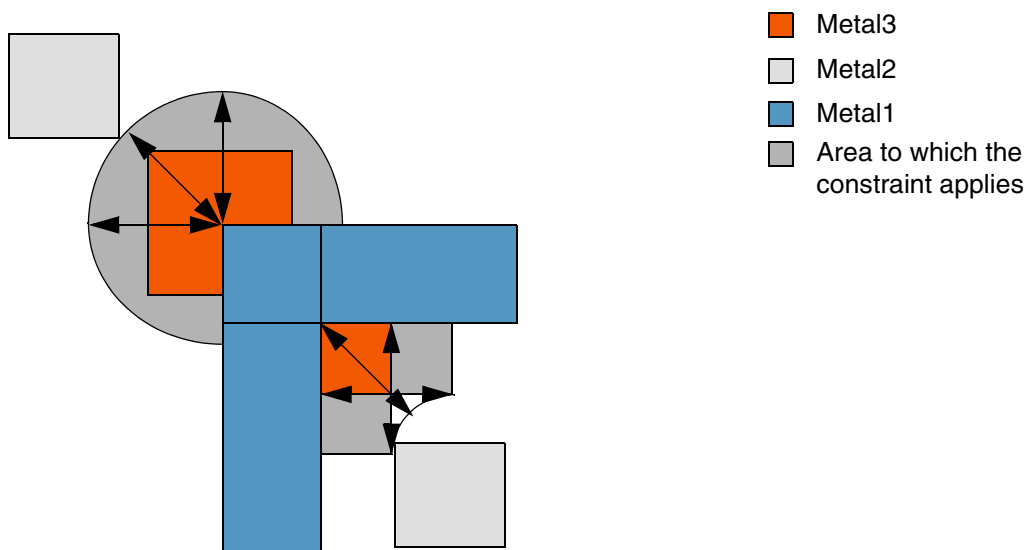
In this second example, in which *layer3* is set to Diffusion, the spacing to the Implant layer must be at least *a* (*spacing*) in the direction of the touching Diffusion layer (that is, in the direction perpendicular to the *layer1* edge overlapped by the *layer3* shape) and *b* (*oppSpacing*) in the direction parallel to this *layer1* edge.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

The constraint also applies to corners where *layer3* and *layer1* touch, as shown in the figure below.



Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer3</i>	The reference layer. The placement of a <i>layer3</i> shape between <i>layer1</i> and <i>layer2</i> shapes determines how the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing in the direction perpendicular to the <i>layer1</i> edge abutted or overlapped by the <i>layer3</i> shape.
<i>f_oppSpacing</i>	The minimum spacing in the direction parallel to the <i>layer1</i> edge abutted or overlapped by the <i>layer3</i> shape.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Parameters

'manhattan

The constraint uses Manhattan distance, which allows a larger spacing at the corners.

By default, the constraint uses Euclidean measurement.

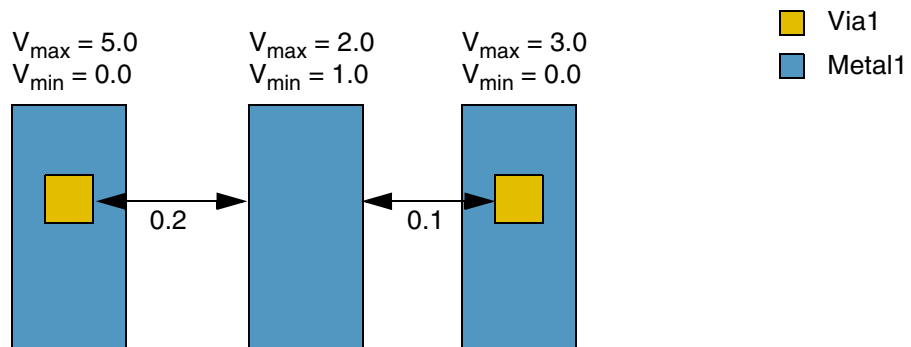
Type: Boolean

minVoltageSpacing (Two layers)

```
spacingTables (
  ( minVoltageSpacing tx_layer1 tx_layer2
    (( "voltage" nil nil )
      ['horizontal | 'vertical]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum spacing between two shapes on different layers when the maximum difference in voltage between the shapes is greater than or equal to the specified voltage. The difference in voltage is calculated as follows:

$$V_{\text{diff}} = \max(V_{\text{max}} \text{ of each shape}) - \min(V_{\text{min}} \text{ of each shape})$$



Shape1 to Shape2

Voltage difference $\max(5,2) - \min(0,1) = 5 - 0 = 5$

Shape2 to Shape3

$\max(2,3) - \min(1,0) = 3 - 0 = 3$

Values

tx_layer1

The first layer on which the constraint is applied.

Type: String (layer name) or Integer (layer number)

tx_layer2

The second layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

"voltage" nil nil

This identifies the index for *table*.

g_table

The format of the *table* row is as follows:

(*f_voltage* *f_distance*)

where,

- *f_voltage* is the voltage difference between two shapes. The voltage must be greater than or equal to this value.
- *f_distance* is the minimum required spacing.

Type: A 1-D table specifying floating-point voltage and spacing values

Parameters

'horizontal' | 'vertical'

(Virtuoso Advanced Node for Layout Only) The direction in which spacing is measured between two cut shapes. If this parameter is not specified, the spacing is measured orthogonally in both directions.

Note: The constraint applies only if the parallel run length between two cut shapes is greater than zero.

Type: Boolean

Examples

- Example 1: minVoltageSpacing with metal layer and cut layer
- Example 2: minVoltageSpacing with two cut layers and spacing direction

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example 1: minVoltageSpacing with metal layer and cut layer

The spacing between wires on a metal layer and via cuts on an adjacent cut layer must be at least:

- 0.065 if the maximum voltage difference between the wire and the via cut is greater than or equal to 0.0V and less than 1.5V
- 0.085 if the maximum voltage difference between the wire and the via cut is greater than or equal to 1.5V and less than 3.3V
- 0.105 if the maximum voltage difference between the wire and the via cut is greater than or equal to 3.3V

```
spacingTables(  
  ( minVoltageSpacing "Metal1" "Via1"  
    (( "voltage" nil nil ))  
    (  
      0.0  0.065  
      1.5  0.085  
      3.3  0.105  
    )  
  )  
) ;spacingTables
```

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

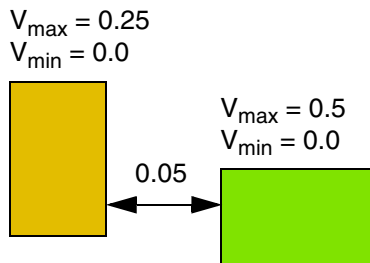
Example 2: minVoltageSpacing with two cut layers and spacing direction

The horizontal spacing between a Via1 cut shape and a Via2 cut shape must be at least:

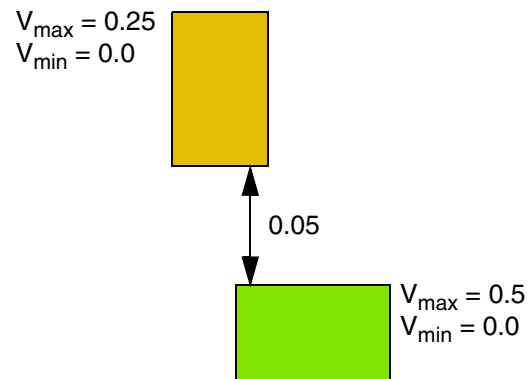
- 0.076 if the maximum voltage difference between the cut shapes is greater than or equal to 0.0V and less than 1.0V
- 0.105 if the maximum voltage difference between the cut shapes is greater than or equal to 1.0V

```
spacingTables(
  ( minVoltageSpacing "Via1" "Via2"
    (( "voltage" nil nil )
      'horizontal
    )
  )
  (
    0.0 0.076
    1.0 0.105
  )
) ;spacingTables
```

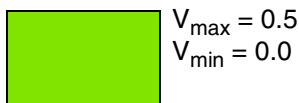
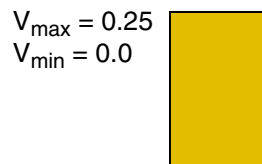
■ Via2
■ Via1



a) FAIL. The voltage difference is 0.5 (greater than 0.0V and less than 1.0V), but the spacing in the horizontal direction is 0.05 (<0.076).



b) The constraint does not apply because the spacing is measured in the vertical direction.



c) The constraint does not apply because the parallel run length between the two shapes is less than zero.

shapeRequiredSpacing

```
orderedSpacings (
    ( shapeRequiredSpacing tx_layer1 tx_layer2
      'horizontal | 'vertical ['twoSides]
      ['widthRanges (g_widthRanges)]
      ['sideExtension f_sideExt]
      (g_ranges)
    )
) ;orderedSpacings

spacingTables (
    ( shapeRequiredSpacing tx_layer1 tx_layer2
      (( "width" nil nil )
        'horizontal | 'vertical ['twoSides]
        ['widthRanges () g_widthRanges)]
        ['sideExtension f_sideExt]
        f_default
      )
      (g_table)
    )
) ;spacingTables
```

Defines the allowed spacing ranges between a shape on *layer1* and a shape on *layer2*. The *layer2* shape must be present along the full parallel run length of the *layer1* shape.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_ranges</i>	The allowed spacing ranges between shapes on different layers. Type: Floating-point values specifying allowed spacing <u>ranges</u>
"width" nil nil	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

g_table The format of a *table* row is as follows:

(f_width (g_ranges))

where, *f_width* is the width of the *layer1* shape and *g_ranges* is the spacing allowed between *layer1* and *layer2* shapes when the width of the *layer1* shape is greater than or equal to the index value.

Type: A 1-D table specifying floating-point width values and the corresponding allowed spacing ranges

Parameters

'horizontal | 'vertical

The direction in which spacing is measured.

'twosides A *layer2* shape must be present on both sides of the *layer1* shape.

'widthRanges (g_widthRanges)

The constraint applies only if the width of the *layer1* shape falls in these ranges.

Type: Floating-point values specifying a range of widths to which the constraint applies

'sideExtension f_sideExt

The ends of the *layer1* shape must be extended by this value in the direction perpendicular to the spacing direction before applying the constraint.

Examples

- Example 1: shapeRequiredSpacing with horizontal, twoSides, and widthRanges
- Example 2: shapeRequiredSpacing with horizontal, widthRanges, and sideExtension

Virtuoso Technology Data Constraints Reference

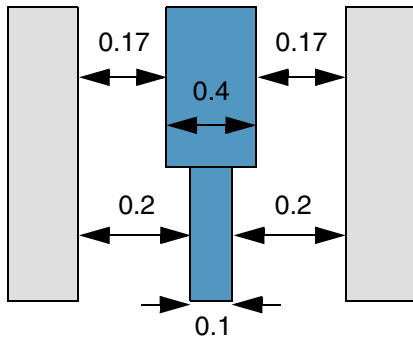
Spacing Constraints (Two Layers)

Example 1: *shapeRequiredSpacing* with *horizontal*, *twoSides*, and *widthRanges*

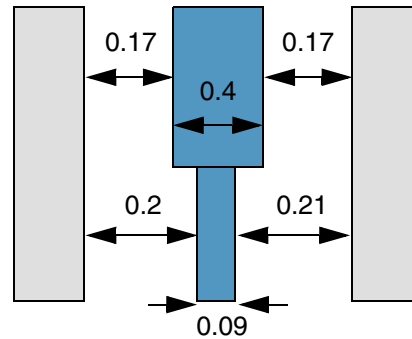
The horizontal spacing between a Metal1 shape and a Metal2 shape must be greater than or equal to 0.1 and less than or equal to 0.2 if Metal2 shapes are present along its full parallel run length on both sides, provided the width of the Metal1 shape is greater than or equal to 0.1 and less than or equal to 0.4.

```
orderedSpacings (
  ( shapeRequiredSpacing "Metal1" "Metal2"
    'horizontal 'twoSides
    'widthRanges ("[0.1 0.4]")
    ("[0.1 0.2]")
  )
) ;orderedSpacings
```

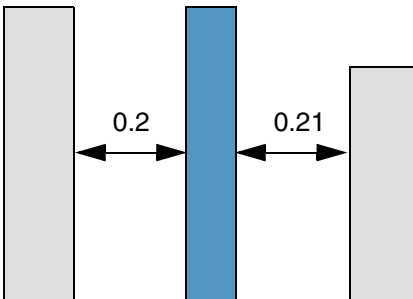
□ Metal2
■ Metal1



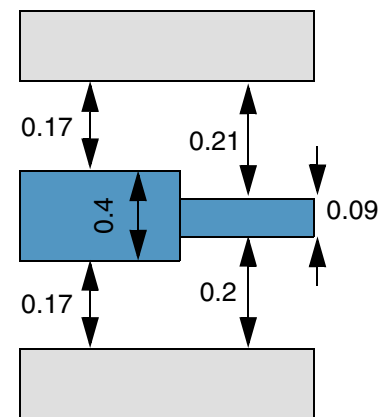
a) PASS. The width of the Metal1 shape is in the range [0.1 0.4] and Metal2 shapes are present along its full parallel run length on both sides. Additionally, the spacing between the Metal1 and Metal2 shapes is in the range [0.1 0.2].



b) FAIL. The spacing between the lower portion of the Metal1 shape and the Metal2 shape is 0.21, which is outside the allowed range of [0.1 0.2].



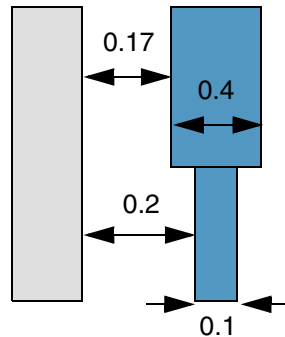
c) FAIL. The Metal2 shape on the right is not present along the full parallel run length of the Metal1 shape.



d) The constraint does not apply because spacing is measured in the vertical direction.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)



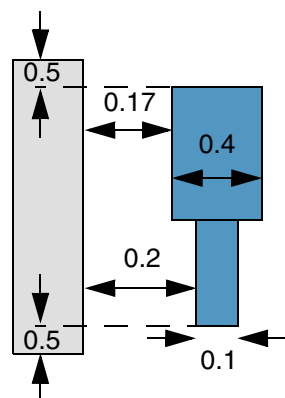
e) FAIL. A Metal2 shape is present on only one side of the Metal1 shape.

Example 2: *shapeRequiredSpacing* with *horizontal*, *widthRanges*, and *sideExtension*

The horizontal spacing between a Metal1 shape and a Metal2 shape must be greater than or equal to 0.1 and less than or equal to 0.2 when the top and bottom edges of the Metal1 shape are extended by 0.05 in the vertical direction, provided the width of the Metal1 shape is greater than or equal to 0.1 and less than or equal to 0.4.

```
orderedSpacings (
  ( shapeRequiredSpacing "Metal1" "Metal2"
    'horizontal
    'widthRanges ("[0.1 0.4]")
    'sideExtension 0.05
    ("[0.1 0.2]")
  )
) ;orderedSpacings
```

□ Metal2
■ Metal1



PASS. The width of the Metal1 shape is in the range [0.1 0.4] and the Metal2 shape is present along its full parallel run length when the top and bottom edges of the Metal1 shape are extended by 0.05 in the vertical direction. Additionally, the spacing between the Metal1 shape and the Metal2 shape is in the range [0.1 0.2].

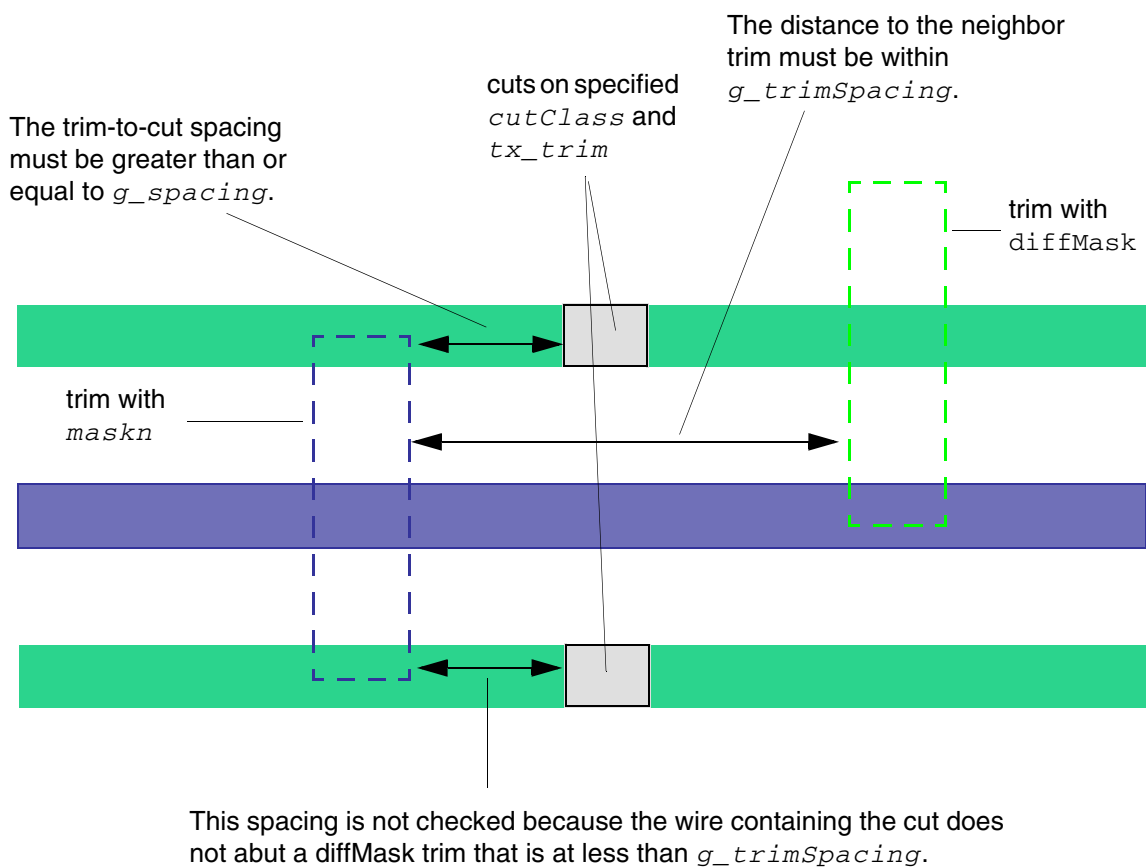
Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

trimMinCutSpacing

```
orderedSpacings (
  ( trimMinCutSpacing tx_trim tx_cut
    ['cutClass {g_width | (g_width g_length) | t_name}]
    ['metalLayer tx_metalLayer
    ['mask1 | 'mask2 | 'mask3]]
    ['trimSpacing g_trimSpacing
      ['otherLayer g_trimLayer | 'diffMask]]
    g_spacing
  )
) ;orderedSpacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the spacing between a trim and a cut.



Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Values

tx_trim tx_cut

The trim and the cut between which the spacing is measured.

Type: String (layer and purpose names) or Integer (layer number)

g_spacing

The spacing between the trim and the cut must be greater than or equal to this value.

Parameters

`'cutClass {g_width | (g_width g_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

`'metalLayer tx_metalLayer`

The constraint applies if the trim and cut are on the specified metal layer.

`'mask1 | 'mask2 | 'mask3`

The constraint applies only to trim shapes on the specified mask.

`'trimSpacing g_trimSpacing ['otherLayer g_trimLayer | 'diffMask]`

The constraint applies only on a trim with a neighbor trim within the given *g_trimSpacing* distance.

Additionally, you can also specify that the spacing applies only to either of the following:

- *otherLayer*: The neighbor trim is on the layer specified by *g_trimLayer*.
- *diffMask*: The neighbor trim is on a different mask.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Example

The spacing between TrimMetal1A and Via1 must be greater than or equal to 0.05 when the shapes are on mask1 and the neighbor trim on layer TrimMetal1B is within 0.07.

```
orderedSpacings (
    (trimMinCutSpacing "TrimMetal1A" "Via1"
        'cutClass "VA"
        'metalLayer "Metal1"
        'mask1
        'trimSpacing 0.07 'otherLayer "TrimMetal1B"
        0.05
    )
) ;orderedSpacings
```

trimMinSpacing (Two layers)

```

spacings (
    ( trimMinSpacing tx_layer1 tx_layer2
      ['prlSpacing (f_spacing1 f_spacing2)]
      ['endToEndSpacing f_endToEndSpacing
        {'horizontal' | 'vertical'}
        ['prl f_prl]
        ['exactAligned f_exactAlignedSpacing]
        ['edgeAligned f_edgeAlignedSpacing]
        ['exceptWidth f_exceptWidth 'layer tx_exceptLayer]
      ]
      ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        ['outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      f_spacing
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Defines the minimum spacing between shapes on the specified trim metal layers.

Values

<i>tx_layer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_layer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between shapes on the two layers must be greater than or equal to this value.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Parameters

`'prlSpacing (f_spacing1 f_spacing2)`

If the parallel run length between the two shapes is zero, the spacing between them must be greater than or equal to *spacing1*, and if the parallel run length between two shapes is greater than zero, the spacing between them must be greater than or equal to *spacing2*.

Otherwise, the spacing between the two shapes must be greater than or equal to *spacing*.

`'endToEndSpacing f_endToEndSpacing`

The 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 *prl*, if specified, or equal to zero.

`'horizontal | 'vertical`

The direction in which end-to-end spacing is measured.

Type: Boolean

`'prl f_prl`

The end-to-end spacing is applied only if the parallel run length between the two shapes is greater than this value, if specified.

`'exactAligned f_exactAlignedSpacing`

The end-to-end spacing must be greater than or equal to this value if the shapes are exactly aligned.

If *prl* is not specified, end-to-end spacing is applied in Euclidian measurement. If *prl* is specified, end-to-end spacing is applied in Manhattan measurement.

`'edgeAligned f_edgeAlignedSpacing`

The end-to-end spacing must be greater than or equal to this value if the top or left edge of one shape is exactly aligned with the bottom or right edge of another shape.

Otherwise, *endToEndSpacing* is applied.

`'exceptWidth f_exceptWidth`

If a shape on *exceptLayer* is between two trim shapes and has width greater than or equal to this value, all end-to-end spacing rules are exempted.

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

`'layer tx_exceptLayer`

The layer on which `'exceptWidth` applies. This is the layer that the trim metal trims.

Type: String (layer name) or Integer (layer number)

`'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)`
`| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)`

Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- `'outsideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Virtuoso Technology Data Constraints Reference

Spacing Constraints (Two Layers)

Via Construction Constraints

This topic includes the following constraints:

- [allowedCutClass](#)
- [bothSidesViaForbiddenSpacingRange](#)
- [cutClassPreference](#)
- [definedCutClassesOnly](#)
- [forbiddenCutClassSpacingRange](#)
- [largeRectViaArrayAllowed](#)
- [maxViaArrayClusterSize](#)
- [maxViaStack](#)
- [mergedViaCornerToCornerSpacing](#)
- [minCutClassSpacing \(One layer\)](#)
- [minCutClassSpacing \(Two layers\)](#)
- [minDirectionalSpacing](#)
- [minEndOfLineCutSpacing](#)
- [minLargeViaArrayCutSpacing](#)
- [minLargeViaArraySpacing](#)
- [minLargeViaArrayWidth](#)
- [minNeighborViaSpacing](#)
- [minNestedViaSpacing](#)
- [minOrthogonalViaSpacing](#)
- [minParallelViaSpacing \(One layer\)](#)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

- minParallelViaSpacing (Two layers plus metal)
- minParallelWithinViaSpacing
- minSameMetalSharedEdgeViaSpacing
- minViaGroupSpacing
- minViaSpacing (One layer)
- minViaSpacing (Two layers)
- multiLayer
- redundantViaSetback
- stackable
- viaEdgeType
- viaGroup
- viaKeepoutMetal
- viaMeshMinCuts
- viaRequiredWithin
- viaSpacing
- viaStackingLimits

allowedCutClass

```
spacingTables(
    ( allowedCutClass tx_cutLayer
      (( "width" nil nil "width" nil nil )
        ['minVoltage f_minVoltage]
        ['maxVoltage f_maxVoltage]
        'cutClass {f_width | (f_width f_length) | t_name}
        ['lowerLayer tx_lowerLayer
          ['lowerMask1 |'lowerMask2 |'lowerMask3]]
        ['upperLayer tx_upperLayer
          ['upperMask1 |'upperMask2 |'upperMask3]]
        ['horizontalLowerDir | 'verticalLowerDir]
        ['horizontalUpperDir | 'verticalUpperDir]
        ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
          ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
          |
          ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
            ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        ]
        x_default
      )
      (g_table)
    )
) ;spacingTables
```

(Virtuoso Advanced Node for Layout Standard) 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.

Note: If all cut classes are allowed on all wire width combinations, the *allowedCutClass* constraint need not be defined.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

tx_cutLayer The cut layer on which the constraint is applied.
Type: String (layer and purpose names) or Integer (layer number)

"width" nil nil "width" nil nil
This identifies the index for *table*.

g_table
The format of a *table* row is as follows:
 $(f\_width1 \ f\_width2) \ x\_value$
where, *f_width1* is the width of the shape on the lower layer and *f_width2* is the width of the shape on the upper layer. *x_value* is 0 if the specified cut class is disallowed for the corresponding shape width combination, and 1 if allowed.
Type: A 2-D table specifying floating-point width values and a 0 or 1 to indicate whether a cut class is disallowed or allowed, respectively

Parameters

'minVoltage *f_minVoltage*
(Virtuoso Advanced Node for Layout Only) The constraint applies only if the voltage on the shape is greater than or equal to this value.

'maxVoltage *f_maxVoltage*
(Virtuoso Advanced Node for Layout Only) The constraint applies only if the voltage on the shape is less than this value.

'cutClass {*f_width* | (*f_width* *f_length*) | *t_name*}
The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).
■ *f_width*: Width
■ *f_length*: Length
■ *t_name*: Name of the cut class

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'lowerLayer tx_lowerLayer`

The metal layer below the cut layer. If not specified, the default stack is used to determine the lower metal layer.

`'lowerMask1 | 'lowerMask2 | 'lowerMask3`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only for the specified color mask shapes of the lower metal layer.

`'upperLayer tx_upperLayer`

The metal layer above the cut layer. If not specified, the default stack is used to determine the upper metal layer.

`'upperMask1 | 'upperMask2 | 'upperMask3`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only for the specified color mask shapes of the upper metal layer.

`'horizontalLowerDir | 'verticalLowerDir`

The direction of the shape on the lower layer. The direction is determined by the wider dimension of the shape.

- `'horizontalLowerDir`: The constraint applies if the shape is horizontal.
- `'verticalLowerDir`: The constraint applies if the shape is vertical.

Type: Boolean

`'horizontalUpperDir | 'verticalUpperDir`

The direction of the shape on the upper layer. The direction is determined by the wider dimension of the shape.

- `'horizontalUpperDir`: The constraint applies if the shape is horizontal.
- `'verticalUpperDir`: The constraint applies if the shape is vertical.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines whether the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
x_default
```

This determines whether cuts are allowed (1) or disallowed (0) by default.

- Set the default table value to 0 and specify the wire width combinations that are allowed with a table value of 1.
- Set the default table value to 1 and specify the wire width combinations that are not allowed with a table value of 0.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Examples

- Example 1: allowedCutClass with cutClass, lowerLayer, upperLayer, horizontalLowerDir, and horizontalUpperDir
- Example 2: allowedCutClass with cutClass, lowerLayer, and upperLayer
- Example 3: allowedCutClass with minVoltage and maxVoltage

Example 1: allowedCutClass with cutClass, lowerLayer, upperLayer, horizontalLowerDir, and horizontalUpperDir

A 0.2x0.4 cut class is allowed between Metal1 and Metal2 wires if the following conditions are met:

- Both wires are horizontal.
- One or both wires are greater than or equal to 0.1 wide.

```
spacingTables(  
  ( allowedCutClass "Via2"  
    (( "width" nil nil "width" nil nil )  
      'cutClass (0.2 0.4)  
      'lowerLayer "Metal1"  
      'upperLayer "Metal2"  
      'horizontalLowerDir  
      'horizontalUpperDir  
    )  
    (  
      (0.0 0.0) 0  
      (0.1 0.1) 1  
    )  
  )  
) ;spacingTables
```

Virtuoso Technology Data Constraints Reference

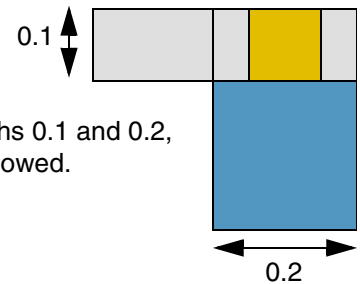
Via Construction Constraints

Example 2: *allowedCutClass* with *cutClass*, *lowerLayer*, and *upperLayer*

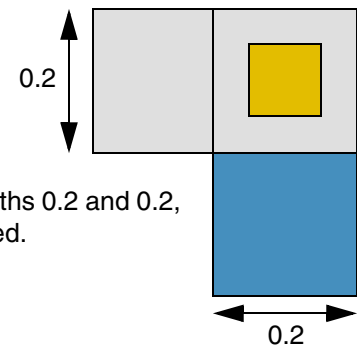
- The V1_small cut class is not allowed between Metal1 and Metal2 wires if one or both wires are greater than or equal to 0.1 wide.
- The V1_large cut class is not allowed between Metal1 and Metal2 wires if one or both wires are 0.1 wide or if both wires are greater than or equal to 0.2 wide.

```
spacingTables(
  ( allowedCutClass "Via1"
    (( "width" nil nil "width" nil nil )
      'cutClass "V1_small"
      'lowerLayer "Metal1"
      'upperLayer "Metal2"
    )
    1
  )
  (
    (0.1 0.1) 0
    (0.1 0.2) 0
    (0.1 0.3) 0
    (0.1 0.4) 0
    (0.2 0.1) 0
    (0.3 0.1) 0
    (0.4 0.1) 0
  )
)
( allowedCutClass "Via1"
  (( "width" nil nil "width" nil nil )
    'cutClass "V1_large"
    'lowerLayer "Metal1"
    'upperLayer "Metal2"
  )
  1
)
(
  (0.1 0.1) 0
  (0.1 0.2) 0
  (0.1 0.3) 0
  (0.1 0.4) 0
  (0.2 0.1) 0
  (0.2 0.2) 0
  (0.3 0.1) 0
  (0.4 0.1) 0
)
) ;spacingTables
```

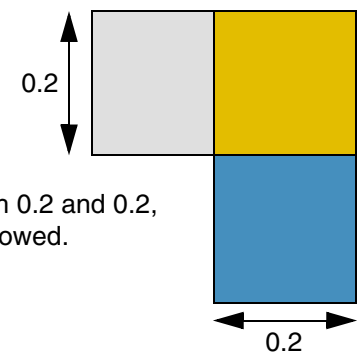
Metal2
 Via1
 Metal1



a) FAIL. For wires with widths 0.1 and 0.2, V1_small cut class is not allowed.



b) PASS. For wires with widths 0.2 and 0.2, V1_small cut class is allowed.



c) FAIL. For wires with width 0.2 and 0.2, V1_large cut class is not allowed.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 3: allowedCutClass with minVoltage and maxVoltage

When voltage is greater than or equal to 0.1 and less than 0.5, the first constraint applies.

When voltage is greater than or equal to 0.5 and less than 0.9, the second constraint applies.

```
( "allowedCutClassCG" nil nil 'and
  spacingTables (
    (allowedCutClass "Via1"
      (("width" nil nil "width" nil nil)
        'minVoltage 0.1
        'maxVoltage 0.5
        'cutClass (0.1 0.2)
        'lowerLayer "Metal1"
        'upperLayer "Metal2"
        1 )
      (
        (0 0.1) 0 )
      )
    )
    (allowedCutClass "Via1"
      (("width" nil nil "width" nil nil)
        'minVoltage 0.5
        'maxVoltage 0.9
        'cutClass (0.08 0.16)
        'lowerLayer "Metal1"
        'upperLayer "Metal2"
        1 )
      (
        (0 0.1) 0 )
      )
    )
  )
);spacingTables
)
```

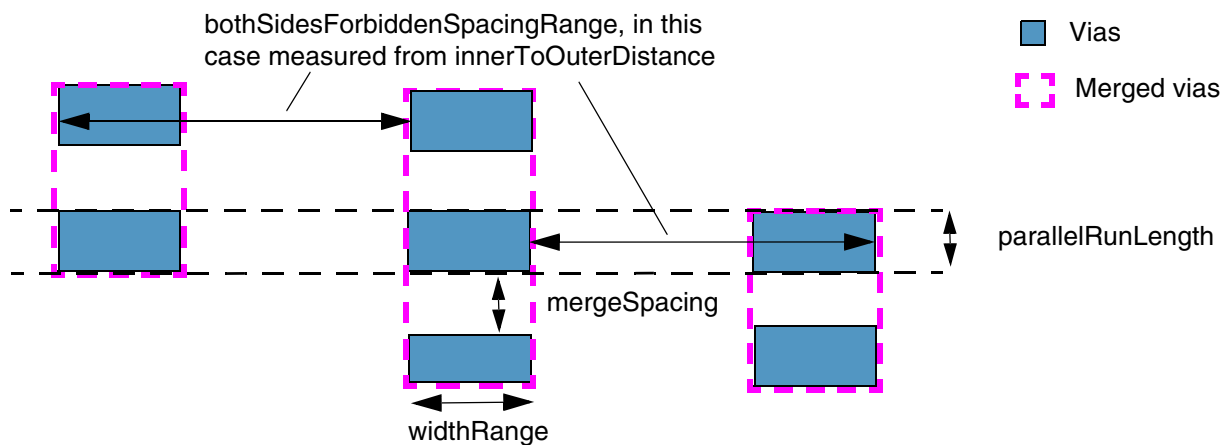
bothSidesViaForbiddenSpacingRange

```

spacings (
  ( bothSidesViaForbiddenSpacingRange tx_layer
    ('vertical | 'horizontal)
    'mergeSpacing g_mergeSpacing
    ['parallelRunLength g_prl]
    | ['sidePRLRange g_sidePRLRange
      ['otherSidePRLRange g_otherSidePRLRange]
    ]
    ['widthRange g_widthRange]
    ['innerToOuterDistance | 'outerToInnerDistance | 'centerToCenter |
    'innerToOuterAndOuterToInner]
    ['sameMask]
    ['otherSideSpacingRange g_otherSideSpacingRange]
    g_range
  )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Defines a forbidden spacing range in a specified direction for three neighboring merged vias.



Values

tx_layer The layer on which the constraint is applied.

 Type: String (layer and purpose names) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

g_range The forbidden spacing range on both sides.
Type: Floating-point values specifying a range of spacings that are not allowed

Parameters

'vertical | 'horizontal

The direction in which shapes should be merged.

- 'vertical: The constraint applies orthogonally in the vertical direction.
- 'horizontal: The constraint applies orthogonally in the horizontal direction.

Type: Boolean

'mergeSpacing *g_mergeSpacing*

Vias that are spaced at less than or equal to this value and are exactly aligned are considered for merging into arrays.

'parallelRunLength *g_prl*

The constraint applies only if the common parallel run length of the merged shapes in the specified direction is more than or equal to *g_prl*.

'sidePRLRange *g_sidePRLRange*

['otherSidePRLRange *g_otherSidePRLRange*]

The constraint applies only if the parallel run length for one side in the specified direction is more than or equal to *g_sidePRLRange*.

When specified, the parallel run length for the other side, *g_otherSidePRLRange*, can also be specified.

'widthRange *g_widthRange*

The constraint applies only if the width of the center shape is in this range. The width is measured in a direction orthogonal to the one specified.

'innerToOuterDistance | 'outerToInnerDistance | 'centerToCenter |
'innerToOuterAndOuterToInner

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The spacing is measured from the inner edge to the outer edge, or conversely, from the outer edge to the inner edge, or from center to center, or from both inner edge to the outer edge and from the outer edge to the inner edge. If not specified, the spacing is measured from edge to edge.

'sameMask

The constraint applies only between shapes on the same mask.

'otherSideSpacingRange *g_otherSideSpacingRange*

The constraint value, *g_range*, determines the spacing for one side of the via and *g_otherSideSpacingRange* determines the spacing for the other side of the via.

Examples

- Example 1: bothSidesViaForbiddenSpacingRange with parallelRunLength
- Example 2: bothSidesViaForbiddenSpacingRange with sidePRLRange and otherSidePRLRange
- Example 3: bothSidesViaForbiddenSpacingRange with outerToInnerDistance
- Example 4: bothSidesViaForbiddenSpacingRange with innerToOuterAndOuterToInner
- Example 5: bothSidesViaForbiddenSpacingRange with otherSideSpacingRange

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

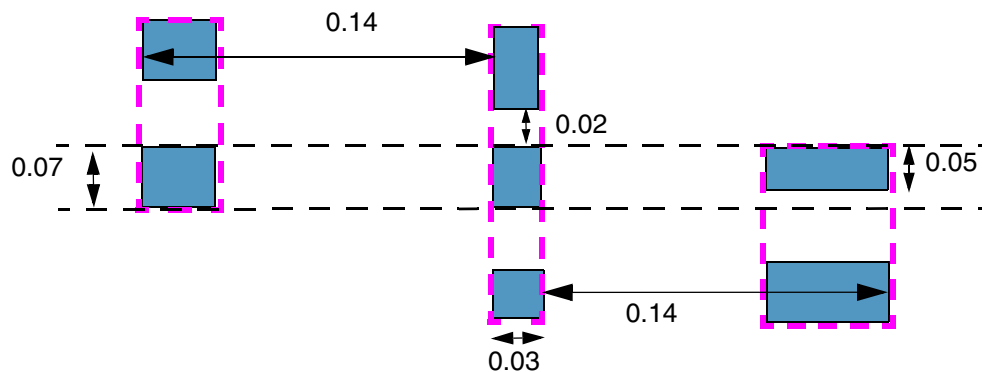
Example 1: bothSidesViaForbiddenSpacingRange with parallelRunLength

The following example illustrates a violation. The common parallel run length is 0.07, which is > 0.06 , and shapes are present in the forbidden spacing range of 0.14.

```
spacings(  
  ( bothSidesViaForbiddenSpacingRange "Via1"  
    'vertical 'mergeSpacing 0.02  
    'parallelRunLength 0.06 'widthRange "<=0.032"  
    'innerToOuterDistance 'sameMask  
    "[0.13 0.15]"  
  )  
)
```

■ Vias

□ Merged vias

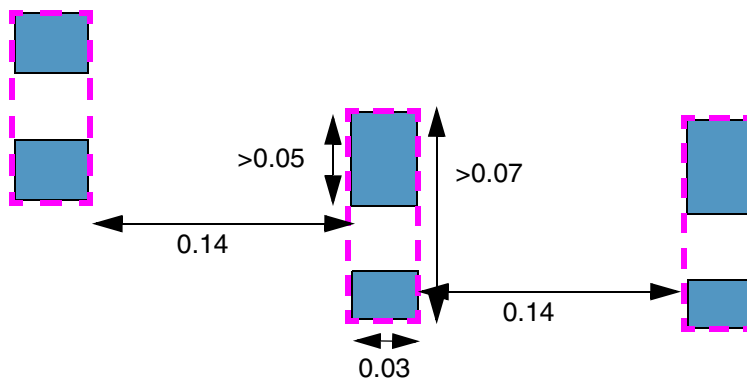


Example 2: *bothSidesViaForbiddenSpacingRange* with *sidePRLRange* and *otherSidePRLRange*

The following example illustrates the use of the `sidePRLRange` and `otherSidePRLRange` parameters and shows a violation. Via triplets are not allowed when one side has parallel run length more than 0.05 and the other side more than 0.07.

```
spacings(
  ( bothSidesViaForbiddenSpacingRange "Via1"
    'vertical 'mergeSpacing 0.12
    'widthRange "<=0.032"
    'sidePRLRange ">0.05"
    'otherSidePRLRange ">0.07"
    'sameMask
    "[0.13 0.15]"
  )
) ;spacings
```

■ Vias
 ■ Merged vias

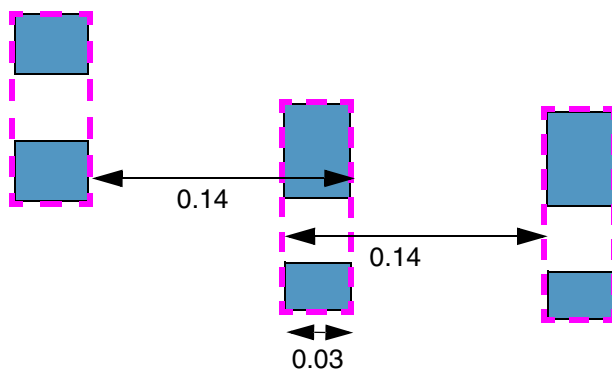


Example 3: bothSidesViaForbiddenSpacingRange with outerToInnerDistance

The following example illustrates the use of the `outerToInnerDistance` parameter and shows a violation. The spacing is measured from the outer edge to the inner edge. The constraint fails because both spacings of 0.14 fall into the forbidden range from 0.13 to 0.15.

```
spacings (
  ( bothSidesViaForbiddenSpacingRange "Via1"
    'vertical 'mergeSpacing 0.12
    'parallelRunLength 0.06
    'widthRange "<=.032"
    'sameMask
    'outerToInnerDistance
    "[0.13 0.15]"
  )
) ;spacings
```

■ Vias
■ Merged via



Virtuoso Technology Data Constraints Reference

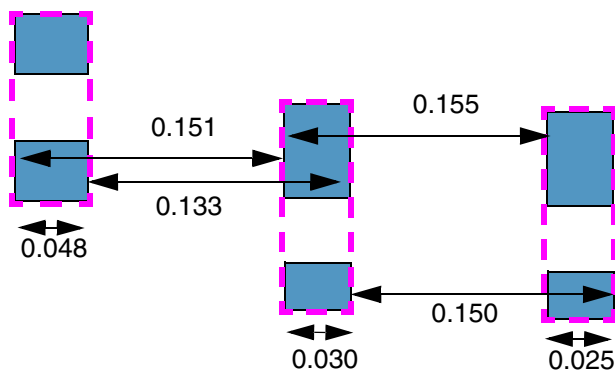
Via Construction Constraints

Example 4: *bothSidesViaForbiddenSpacingRange* with *innerToOuterAndOuterToInner*

The following example illustrates the use of the `innerToOuterAndOuterToInner` parameter and shows a violation. The spacing is measured inner edge to the outer edge and from the outer edge to the inner edge.

```
spacings (
  ( bothSidesViaForbiddenSpacingRange "Via1"
    'vertical 'mergeSpacing 0.12
    'parallelRunLength 0.06
    'widthRange "<=.032"
    'sameMask
    'innerToOuterAndOuterToInner
    "[0.13 0.147]"
  )
) ;spacings
```

■ Vias
■ Merged vias



FAIL. The spacing of 0.133 falls inside the forbidden spacing range.

Virtuoso Technology Data Constraints Reference

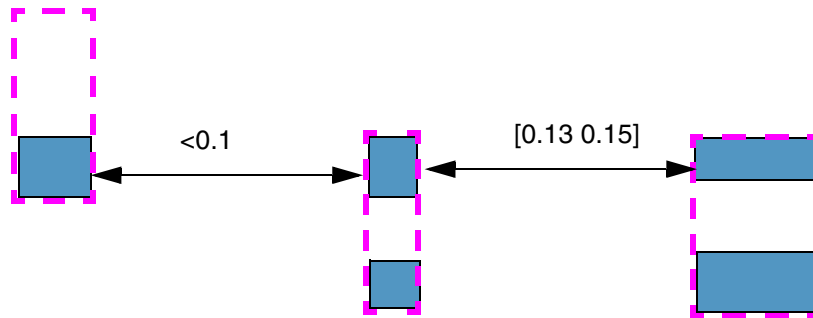
Via Construction Constraints

Example 5: *bothSidesViaForbiddenSpacingRange* with *otherSideSpacingRange*

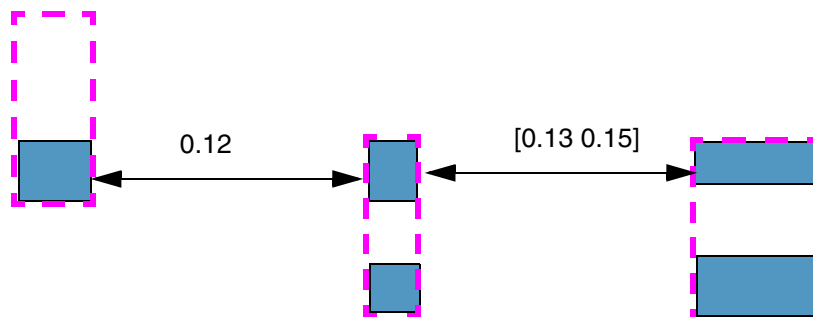
The following example illustrates the use of the `otherSideSpacingRange` parameter in two scenarios.

```
spacings(  
  ( bothSidesViaForbiddenSpacingRange "Via1"  
    'vertical 'mergeSpacing 0.12  
    'parallelRunLength 0.06 'widthRange "<=0.032"  
    'sameMask  
    'otherSideSpacingRange "<0.1"  
    "[0.13 0.15]"  
  )  
) ;spacings
```

■ Vias
■ Merged vias



This is a violation because vias triplets are not allowed when the spacing on one side is less than 0.1 and the spacing on the other side is in the range 0.13 to 0.15.



The constraint passes because the vias triplets do not have spacing less than 0.1 on one side although the spacing on the other side is in the range 0.13 to 0.15.

checkOddViaLoop

```
spacings (
    ( checkOddViaLoop tx_cutLayer
      'horizontal | 'vertical
      'mergeSpacing f_mergeSpacing
      'horizontalSpacing f_hSpacing
      'verticalSpacing f_vSpacing
      ['count x_count]
      ['prl f_prl]
    )
) ;spacings
```

(Virtuoso Layout Suite EXL and higher tiers) Reports a violation when a via loop contains an odd number of vias.

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

<i>'horizontal 'vertical</i>	The direction in which vias should be merged. ■ <i>'horizontal</i>: The constraint applies orthogonally in the horizontal direction. ■ <i>'vertical</i>: The constraint applies orthogonally in the vertical direction. Type: Boolean
<i>'mergeSpacing f_mergeSpacing</i>	Vias that are spaced at less than or equal to this value and are exactly aligned are considered for merging.
<i>'horizontalSpacing f_hSpacing</i>	Vias are part of a loop when the horizontal spacing between two vias is less than or equal to <i>f_hSpacing</i> .

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'verticalSpacing *f_vSpacing*

Vias are part of a loop when the vertical spacing between two vias is less than or equal to *f_vSpacing*.

'count *x_count*

Loop is detected in the region created around the via in horizontal direction using *x_pitch* and *horizontalSpacing* multiplied by the *x_count* value and in vertical direction using *y_pitch* and *verticalSpacing* multiplied by the *x_count* value. The smallest cutClass dimensions are used for *x_pitch* and *y_pitch*.

'prl *f_prl*

The constraint is applicable only if the parallel run length between two vias is greater than or equal to the value of *f_prl*.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

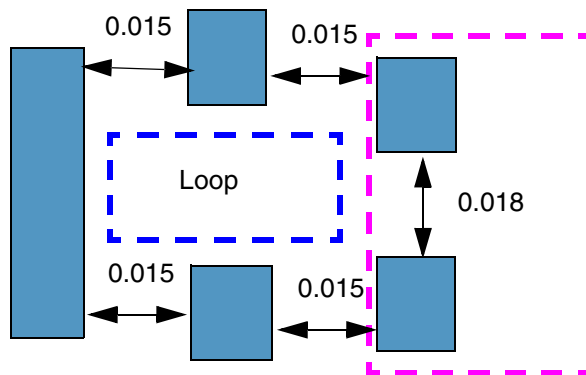
Example

```

spacings (
  (checkOddViaLoop "Via0"
    'vertical
    'mergeSpacing 0.018
    'horizontalSpacing 0.016
    'verticalSpacing 0.021
    'count 12
    'prl 0.011
  )
) ; spacings

```

■ Via0
 ■ Merged vias



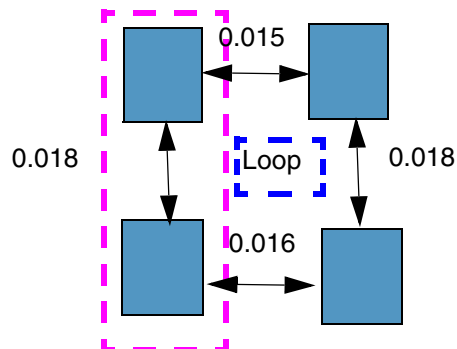
a) PASS. This is a four vias loop. The rightmost vias are merged. The merge direction is specified as vertical, so horizontally aligned vias are not merged.

```

spacings (
  (checkOddViaLoop "Via0"
    'vertical
    'mergeSpacing 0.018
    'horizontalSpacing 0.016
    'verticalSpacing 0.021
    'count 12
    'prl 0.011
  )
) ; spacings

```

■ Via0
 ■ Merged vias



b) FAIL. This is a three vias loop. The leftmost vias are merged but the rightmost vias are not aligned and are therefore not merged.

Virtuoso Technology Data Constraints Reference

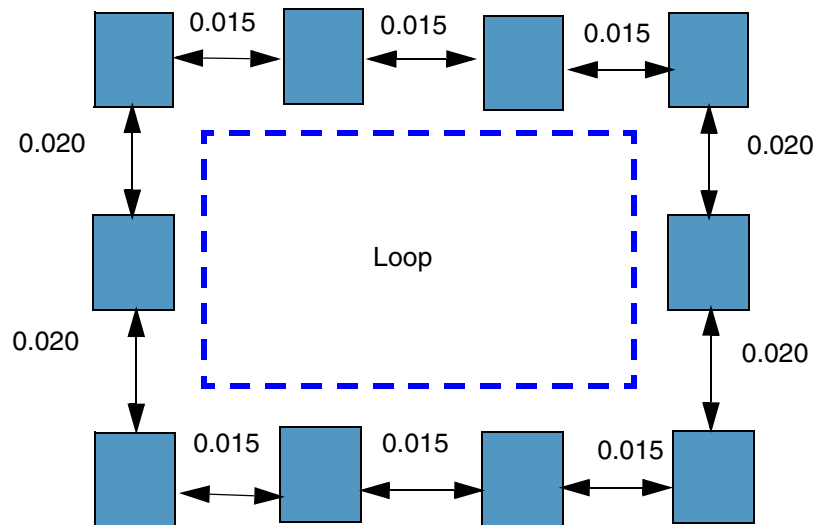
Via Construction Constraints

```

spacings (
  (checkOddViaLoop "Via0"
    'vertical
    'mergeSpacing 0.018
    'horizontalSpacing 0.016
    'verticalSpacing 0.021
    'count 12
    'prl 0.011
  )
) ;spacings

```

 Via0
 Merged vias



c) PASS. This is a ten vias loop. The merge direction is vertical, so horizontally aligned vias are not merged. All vertically aligned vias are spaced > 0.018 so they are also not merged.

Virtuoso Technology Data Constraints Reference

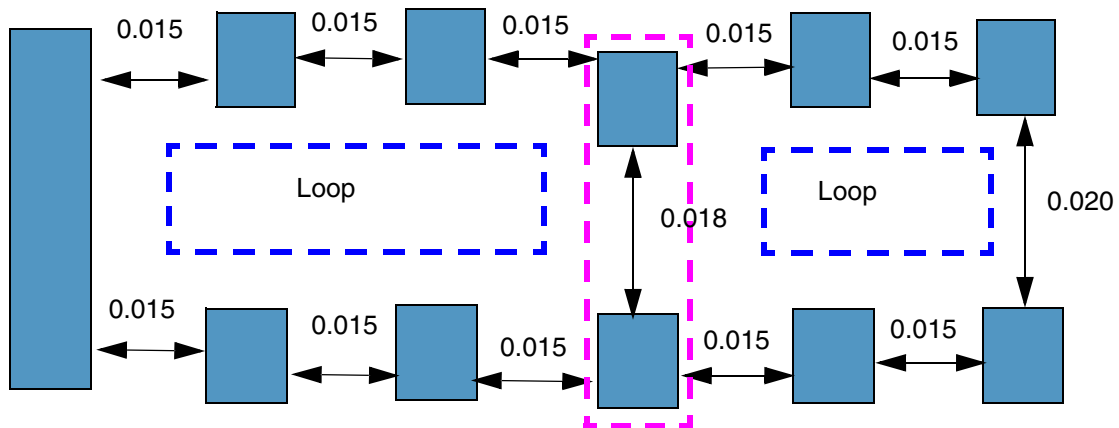
Via Construction Constraints

```

spacings (
  (checkOddViaLoop "Via0"
    'vertical
    'mergeSpacing      0.018
    'horizontalSpacing  0.016
    'verticalSpacing    0.021
    'count              12
    'prl                0.011
  )
) ;spacings

```

Via0
 Merged vias



d) FAIL. After the two vias in the center are merged, the loop on the right has five vias, which is an odd number. The loop on the left is not a violation because it has six vias, which is an even number.

cutClassPreference

```
spacings (
    ( cutClassPreference tx_cutLayer
      ['lowerLayer tx_lowerLayer]
      ['upperLayer tx_upperLayer]
      ['horizontalLowerDir | 'verticalLowerDir]
      ['horizontalUpperDir | 'verticalUpperDir]
      ['lowerWidthRange g_lowerWidthRange]
      ['upperWidthRange g_upperWidthRange]
      l_cutClasses
    )
) ;spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies an ordered list of cut classes. This constraint can be used in conjunction with the `allowedCutClass` constraint to implement the preferred order. It can be used to limit cut classes or change the preferred order of cut classes in constraint groups.

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>l_cutClasses</i>	A list of cut classes in a preferred order. Each cut class is specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class

Parameters

`'lowerLayer tx_lowerLayer`

The metal layer below the cut layer. If not specified, the default stack is used to determine the lower metal layer.

`'upperLayer tx_upperLayer`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The metal layer above the cut layer. If not specified, the default stack is used to determine the upper metal layer.

`'horizontalLowerDir` | `'verticalLowerDir`

The direction of the shape on the lower layer. The direction is determined by the wider dimension of the shape.

- `'horizontalLowerDir`: The constraint applies if the shape is horizontal.
- `'verticalLowerDir`: The constraint applies if the shape is vertical.

Type: Boolean

`'horizontalUpperDir` | `'verticalUpperDir`

The direction of the shape on the upper layer. The direction is determined by the wider dimension of the shape.

- `'horizontalUpperDir`: The constraint applies if the shape is horizontal.
- `'verticalUpperDir`: The constraint applies if the shape is vertical.

Type: Boolean

`'lowerWidthRange` *`g_lowerWidthRange`*

The constraint applies if the width of the lower metal layer is in this range.

`'upperWidthRange` *`g_upperWidthRange`*

The constraint applies if the width of the upper metal layer is in this range.

Example

The square 0.034x0.034 cut class has preference over the rectangular 0.07x0.034 cut class.

```
spacings (
  (cutClassPreference "Vial"
    ((0.034 0.034) (0.07 0.034))
  )
) ;spacings
```

definedCutClassesOnly

```
spacings (
    ( definedCutClassesOnly tx_layer )
) ;spacings
```

Requires that all cut shapes on the specified layer must match the cutClasses defined for the layer.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Example

For layer Via1, only those cut shapes for which a cut class exists are legal.

```
spacings (
    ( definedCutClassesOnly "Via1" )
) ;spacings
```

forbiddenCutClassSpacingRange

```
spacings (
  ( forbiddenCutClassSpacingRange tx_layer
    'cutClass {f_width | (f_width f_length) | t_name}
    ['layer tx_metalLayer
      ['mask1 | 'mask2 | 'mask3]
      ['layerWidth f_layerWidth]
      'paraLength f_paraLength 'within f_within
    ]
    ['sameMask]
    ['shortEdge | 'longEdge] ['vertical | 'horizontal]
    ['cutClassPrl g_cutClassPrl | 'allCuts]
    g_range
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies that the spacing between two cut shapes of the specified cut class on the specified layer cannot fall in the specified range.

In some processes, this spacing restriction applies in addition to the normal cut-class-specific rules. In some cases, this additional spacing restriction applies only to the spacing between the short or long edges of two cut shapes of a rectangular cut class with parallel run length greater than zero, or between an edge of a rectangular cut shape and any edge of a cut shape of the specified cut class if the parallel run length between the two shapes is greater than the specified value.

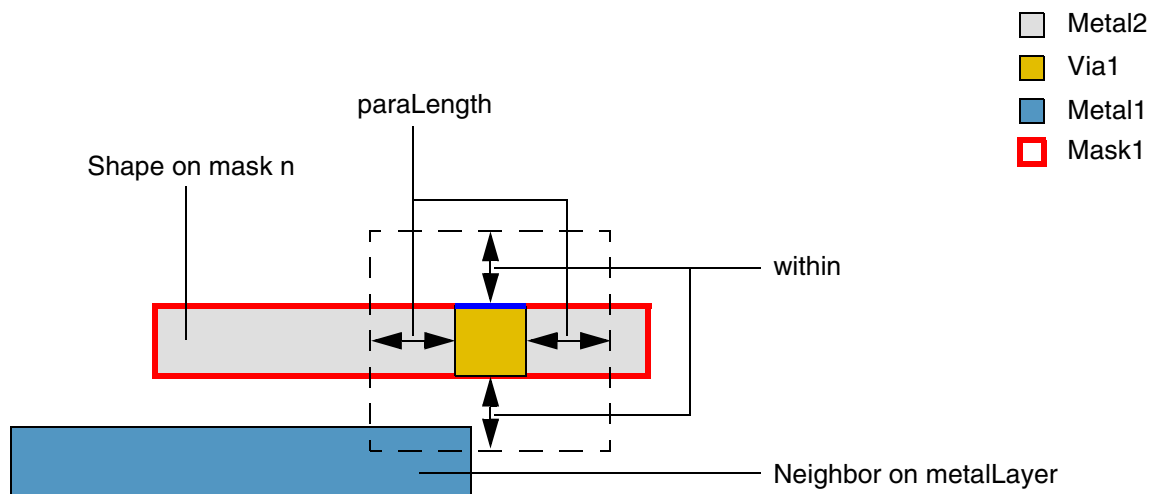
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

If a neighboring metal shape is present on *metalLayer* on one side of the cut shape of the specified cut class, the constraint applies on the other side of the cut shape (the blue edge shown in the figure below) provided the following conditions are met:

- The parallel run length between the two metal shapes—the metal shape that encloses the cut shape and the metal neighbor—is greater than or equal to *paraLength*.
- The distance between the cut shape and the metal neighbor is less than *within*.

Optionally, if the mask type is specified, the constraint applies only to cut shapes enclosed by metal shapes on that mask.



Values

tx_layer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

g_range

The forbidden spacing range is specified using the format:

$[f\_min \ f\_max]$

where, $f\_min \leq \text{spacing} \leq f\_max$

Type: Floating-point values specifying a range of spacings that are not allowed

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

`'cutClass {f_width | (f_width f_length) | t_name}`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'layer tx_metalLayer`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the cut shape has a metal neighbor on this layer on one side.

Type: String (layer name) or Integer (layer number)

`'mask1 | 'mask2 | 'mask3`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the cut shape is enclosed by a metal shape on the specified mask and has a metal neighbor on one side.

Type: Boolean

`'layerWidth f_layerWidth`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the width of the metal neighbor is less than or equal to this value.

`'paraLength f_paraLength`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the parallel run length between the two metal shapes—the metal shape that encloses the cut shape and the metal neighbor—is greater than or equal to this value.

`'within f_within`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the distance between the cut shape and the metal neighbor is less than this value.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'sameMask

(Virtuoso Advanced Node for Layout Only) The constraint applies only between same-mask cut shapes of the specified cut class.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'shortEdge | 'longEdge | 'vertical | 'horizontal

The edges to which the constraint applies.

- 'shortEdge: The constraint applies only between the short edges of rectangular cut shapes if the two cut shapes have parallel run length greater than zero.
- 'longEdge: (Virtuoso Advanced Node for Layout Only)
The constraint applies only between the long edges of rectangular cut shapes if the two cut shapes have parallel run length greater than zero.
- 'vertical: (Virtuoso Advanced Node for Layout Only)
The constraint applies only to edges in the vertical direction.
- 'horizontal: (Virtuoso Advanced Node for Layout Only)
The constraint applies only to edges in the horizontal direction.

Type: Boolean

'cutClassPr1 *g_cutClassPr1*

The constraint applies between a rectangular cut shape and another cut shape of the cut class specified with this parameter if the two cut shapes have parallel run length greater than this value.

If the specified parallel run length is negative, the edge of the rectangular cut shape is extended in both directions by the absolute value of the specified parallel run length, and the constraint applies between the extended edge and any neighboring cut edge of the specified cut class.

The value of the parameter is defined as follows:

(*f_width f_length f_pr1 ...*)

where,

- *f_width* is the width of the cut class.
- *f_length* is the length of the cut class.
- *f_pr1* is the parallel run length to a cut shape with these dimensions.

Type: An array specifying floating-point values

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'allCuts

(Virtuoso Advanced Node for Layout Only) The constraint applies between a cut shape of the specified cut class and a cut shape of any other cut class.

Type: Boolean

Examples

- Example 1: forbiddenCutClassSpacingRange with shortEdge
- Example 2: forbiddenCutClassSpacingRange with cutClassPrl

Virtuoso Technology Data Constraints Reference

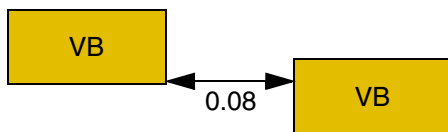
Via Construction Constraints

Example 1: *forbiddenCutClassSpacingRange* with *shortEdge*

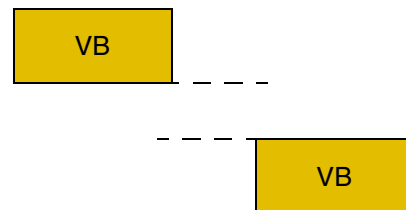
The spacing between the short edges of two VB cut shapes on layer Via1 must not be greater than or equal to 0.07 and less than or equal to 0.09 if the cut shapes have parallel run length greater than zero.

```
spacings (
  ( forbiddenCutClassSpacingRange "Via1"
    'cutClass ("VB")
    'shortEdge
    "[0.07 0.09]"
  )
) ;spacings
```

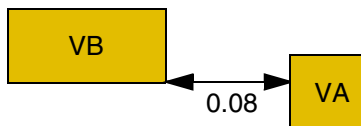
 Via1



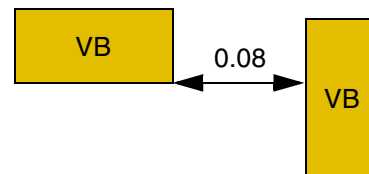
a) FAIL. The spacing between two short edges is 0.08, which lies in the forbidden spacing range of [0.07 0.09].



b) The constraint does not apply because the cut shapes have parallel length less than zero. A violation would occur if 'shortEdge' was not specified.



c) The constraint does not apply because the cut shapes belong to two different cut classes, VA and VB. The constraint applies only to VB cuts.



d) The constraint does not apply because there is no parallel run length between the short edges of the two cut shapes. A violation would occur if 'shortEdge' was not specified.

Virtuoso Technology Data Constraints Reference

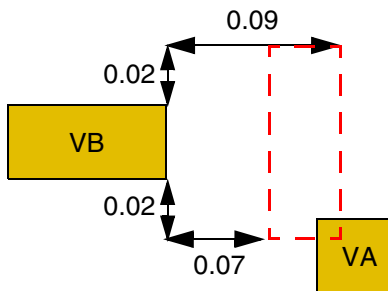
Via Construction Constraints

Example 2: *forbiddenCutClassSpacingRange* with *cutClassPr1*

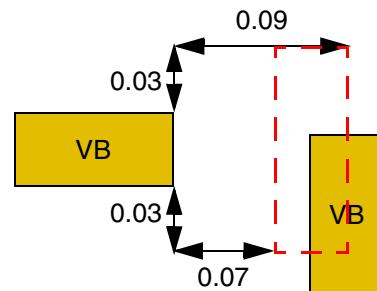
The spacing between the short edge of a VB cut shape to any edge of a VA or VB cut on layer Via1 must not be greater than or equal to 0.07 and less than or equal to 0.09. The spacing applies when the parallel run length between VA and VB cut shapes is greater than -0.02 and between two VB shapes is greater than -0.03.

```
spacings(  
  ( forbiddenCutClassSpacingRange "Via1"  
    'cutClass ("VB")  
    'shortEdge  
    'cutClassPr1 (0.3 0.3 -0.02 0.3 0.6 -0.03)  
    "[0.07 0.09]"  
  )  
) ;spacings
```

 Via1



a) FAIL. The VA cut shape overlaps the forbidden spacing region (indicated by the red dotted line).



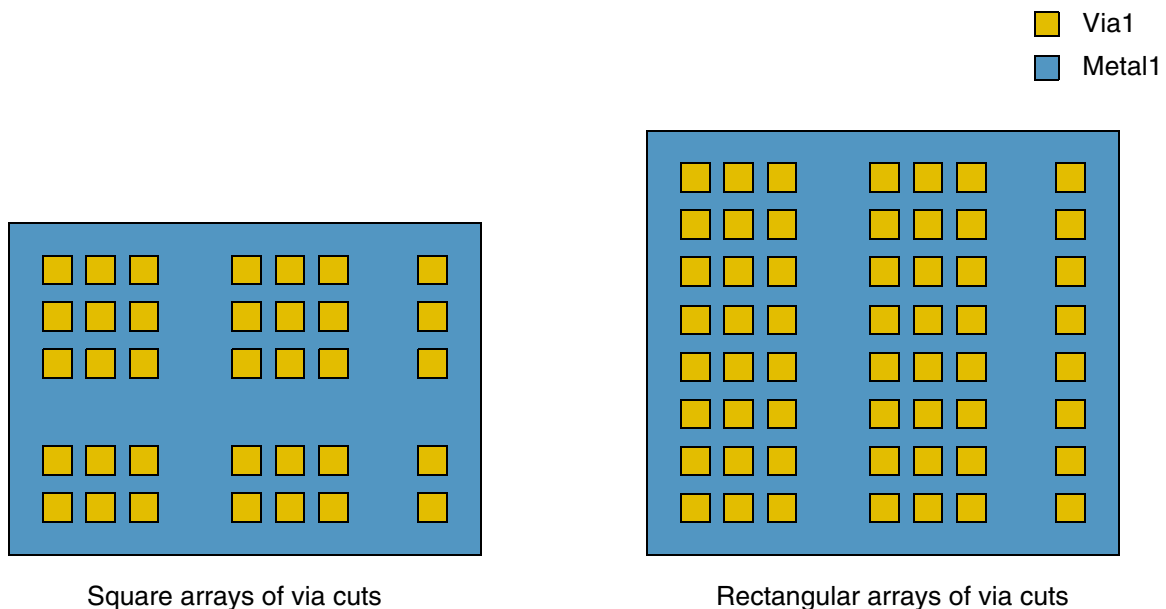
b) FAIL. The VB cut shape on the right overlaps the forbidden spacing region (indicated by the red dotted line). The constraint would not apply if 'cutClassPr1 was not specified.

largeRectViaArrayAllowed

```
spacings (  
  ( largeRectViaArrayAllowed tx_cutLayer  
    ['cutClass {f_width | (f_width f_length) | t_name}]  
    ['paraOverlap]  
    {t | nil}  
  )  
) ;spacings
```

Specifies whether a rectangular array of via cuts is allowed. By default, only square arrays of via cuts are supported. If the constraint allows rectangular arrays, both square and rectangular arrays are supported.

The via can be cut off at the end of a wire, as shown in the figure below, which results in incomplete arrays. However, this is not considered a violation.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<code>tx_cutLayer</code>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>t nil</code>	If <code>t</code> , rectangular arrays are allowed; if <code>nil</code> , only square arrays are allowed.

Parameters

<code>'cutClass {f_width (f_width f_length) t_name}</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <code>f_width</code>: Width ■ <code>f_length</code>: Length ■ <code>t_name</code>: Name of the cut class
<code>'paraOverlap</code>	The constraint applies only if the via cut arrays have a non-zero parallel overlap. Type: Boolean

Example

Rectangular arrays of via cuts are allowed on layer Cut1, but not on layer Cut2.

```
spacings (
  ( largeRectViaArrayAllowed "Cut1" t )
  ( largeRectViaArrayAllowed "Cut2" nil )
) ;spacings
```

maxViaArrayClusterSize

```

spacings (
    (maxViaArrayClusterSize tx_layer
        'horizontal | 'vertical
        'mergeSpacing f_mergeSpacing
        {'clusterSpacing f_clusterSpacing
            ['clusterSpacingRange g_clusterSpacingRange
                ['lengthRange g_lengthRange ['checkLengthOfAnyArray]]
                ['clusterAny | 'clusterHorizontal | 'clusterVertical
                    ['neighborDistanceRange g_neighborDistanceRange
                        ['maxFacingEdgeNeighbors g_NumNeighbors
                            ['checkEachSideNeighborsSeparately]
                    ]
                ]
            ]
        }
        ['layer g_layer ['colorMasks (mask1 mask2 ... maskn)]]
        ['enclosingLayer tx_enclosingLayer
            ['enclosureValue f_enclosureValue]
        ]
        ['clusterLength f_clusterLength]
        ['exceptExactAligned]
        ['sameMask]
        ['minNumCutsInArray x_minNumCutsInArray]
        ['maxNumCutsInArray x_maxNumCutsInArray]
        ['minClusterSize x_clusterSize]
        x_numViaArrays
    )
) ;spacings

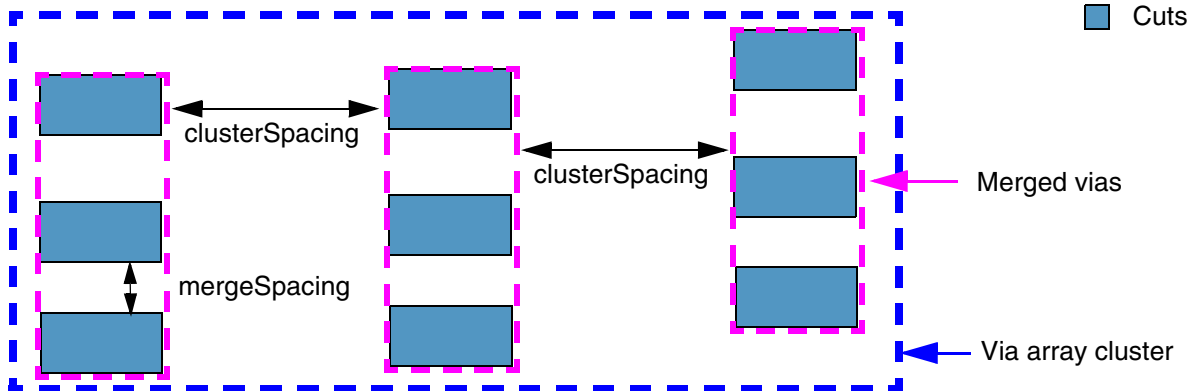
```

(Virtuoso Advanced Node for Layout Only) Specifies the maximum number of via arrays that are allowed in a cluster.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The following example shows three vertically aligned and merged via arrays in a cluster.



Values

<code>tx_layer</code>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>x_numViaArrays</code>	The maximum number of via arrays that are allowed in a cluster.

Parameters

<code>'horizontal 'vertical</code>	The direction in which the vias should be merged into arrays. Type: Boolean
<code>'mergeSpacing f_mergeSpacing</code>	Vias that are spaced at less than or equal to this value and are exactly aligned are considered for merging into arrays.
<code>'clusterSpacing f_clusterSpacing</code>	Via arrays that are spaced at less than this value are considered for clustering.
<code>'clusterSpacingRange g_clusterSpacingRange</code>	Shapes in this spacing range are considered for clusters. This parameter is an alternative to the <code>clusterSpacing</code> parameter.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'lengthRange g_lengthRange ['checkLengthOfAnyArray]`

Specified with `clusterSpacingRange`. Only shapes in this range are considered. It is measured in the merge direction.

If `checkLengthOfAnyArray` is also specified, the constraint applies if the length of any one of the via arrays is within `g_lengthRange`.

`'clusterAny | 'clusterHorizontal | 'clusterVertical`

Specified with `clusterSpacingRange`. It specifies the direction in which cluster distance should be measured. It can have one of these values: `clusterAny` (default), `clusterHorizontal`, or `clusterVertical`.

Type: Boolean

`'neighborDistanceRange g_neighborDistanceRange`

Specified when the cluster direction is specified as `clusterHorizontal` or `clusterVertical`. Neighbors are considered within this range in a direction orthogonal to the cluster direction.

`'maxFacingEdgeNeighbors g_NumNeighbors`

Specified with `neighborDistanceRange` to indicate the maximum number of facing edge neighbors within the specified `neighborDistanceRange` for a cluster.

`'checkEachSideNeighborsSeparately`

Specified with `maxFacingEdgeNeighbors` to indicate that neighbors on each side must be checked.

`'prl f_prl`

Specified with `clusterSpacingRange`. The constraint will be applicable only if all shapes in the cluster have parallel run length within this range.

`'sameMask`

The constraint applies only between vias on the same mask.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'layer g_layer ['colorMasks (mask1 mask2 ... maskn)']`

The constraint applies only for cuts on the specified layer. The via can be either intersecting or enclosed within the metal.

If the `colorMasks` parameter is also specified, the constraint applies selectively to cuts on the specified color masks. Masks are specified using integer values starting from 1 representing `mask1`.

`'enclosingLayer tx_enclosingLayer`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies for the metal layer for which enclosures are checked on all sides of the via.

`'enclosureValue f_enclosureValue`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies when enclosure of vias on the layer specified in the `enclosingLayer` parameter in any direction is less than this value.

`'clusterLength f_clusterLength`

The constraint applies only if the length of the cluster is greater than this value.

`'exceptExactAligned`

The constraint applies only if the merged vias are not exactly aligned.

Type: Boolean

`'minNumCutsInArray x_minNumCutsInArray`

The constraint applies only if at least one array in the cluster has at least this number of cuts.

`'maxNumCutsInArray x_maxNumCutsInArray`

The constraint shows a violation if any array in the cluster has more than this number of cuts.

`'minClusterSize x_clusterSize`

The constraint applies only when the number of via arrays in a cluster is more than or equal to this value.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Examples

- Example 1: maxViaArrayClusterSize with numViaArrays and minNumCutsInArray
- Example 2: maxViaArrayClusterSize with clusterSpacingRange
- Example 3: maxViaArrayClusterSize with neighborDistanceRange and maxFacingEdgeNeighbors
- Example 4: maxViaArrayClusterSize with neighborDistanceRange, maxFacingEdgeNeighbors, and checkEachSideNeighborsSeparately
- Example 5: maxViaArrayClusterSize with checkLengthOfAnyArray
- Example 6: maxViaArrayClusterSize with layer and colorMasks
- Example 7: maxViaArrayClusterSize with clusterLength
- Example 8: maxViaArrayClusterSize with maxNumCutsInArray and minClusterSize

Example 1: maxViaArrayClusterSize with numViaArrays and minNumCutsInArray

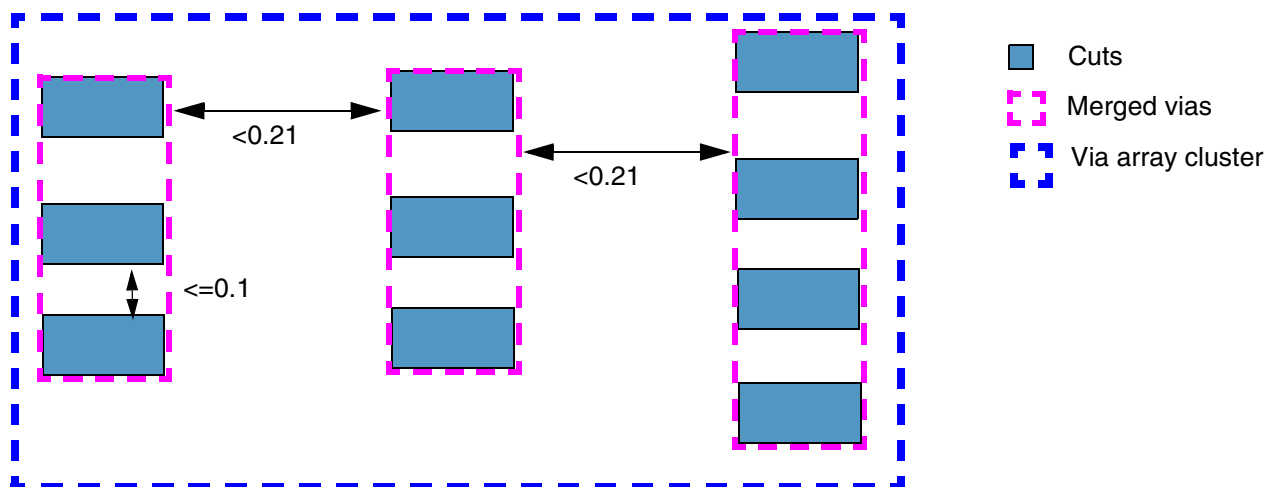
Consider the following constraint:

```
spacings (
  ( maxViaArrayClusterSize "Vial"
    'vertical
    'mergeSpacing 0.11
    'clusterSpacing 0.21
    'exceptExactAligned
    'sameMask
    'minNumCutsInArray 4
    2
  )
) ;spacings
```

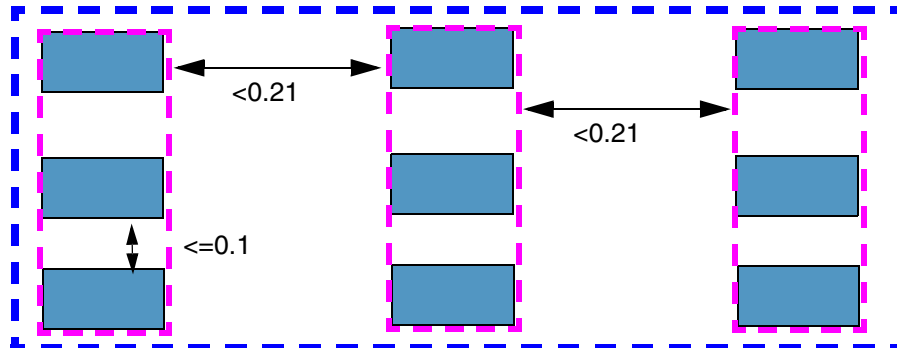
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

A couple of sample scenarios follow:



a. The via array cluster violates the constraint because it has more than 2 merged via arrays whereas the value of $x\_numViaArrays$ is 2. In case of $x\_minNumCutsInArray$, the constraint is applicable because one via array has 4 elements.



b. The constraint does not apply because all via arrays are exactly aligned and none of the via arrays has more than or equal to 4 elements.

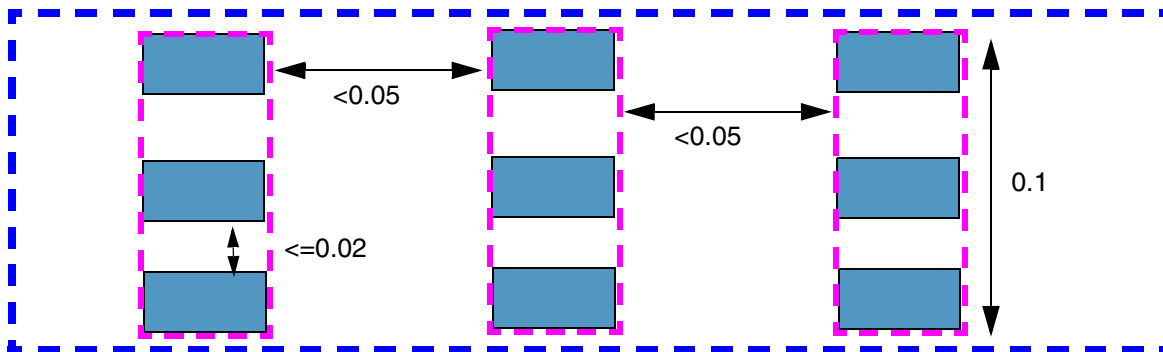
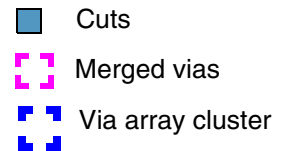
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 2: *maxViaArrayClusterSize* with *clusterSpacingRange*

Vertically aligned vias that are within 0.02 are merged. Merged vias that are at least 0.1 in length and at a distance of less than 0.05 in the horizontal direction are considered as clustered. The constraint is met if there are no more than 2 merged vias in the cluster.

```
spacings(  
  ( (maxViaArrayClusterSize "Vial"  
    'vertical  
    'mergeSpacing "0.02"  
    'clusterSpacingRange "<0.05"  
    'lengthRange ">=0.1"  
    'clusterHorizontal  
      2  
    )  
  )  
); spacings
```



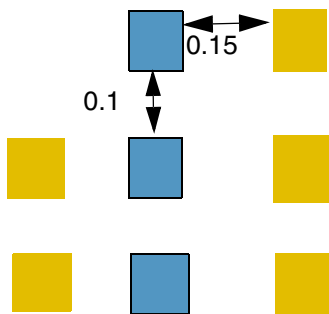
Fail. The constraint applies because there are aligned cuts within 0.02 that are merged in the vertical direction. The constraint is violated because the cluster has more than 2 merged vias but the maximum number of allowed merged vias is 2.

Example 3: *maxViaArrayClusterSize* with *neighborDistanceRange* and *maxFacingEdgeNeighbors*

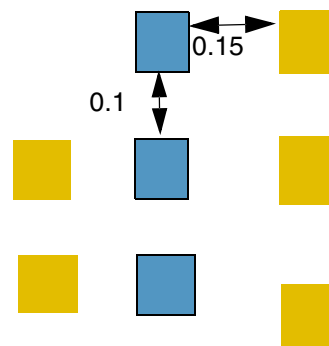
The maximum cluster size is 3, and it can have a maximum of 4 neighbors.

```
spacings (
  ( (maxViaArrayClusterSize "Via1"
    'horizontal
    'mergeSpacing 0
    'clusterSpacingRange "<=0.1"
    'clusterVertical
    'neighborDistanceRange "<=0.15"
    'maxFacingEdgeNeighbors 4
    2
  )
) ;spacings
```

■ Vias in cluster
■ Neighboring vias



a. Fail: The maximum cluster size is 3. It can have only 4 neighbors but has 5.



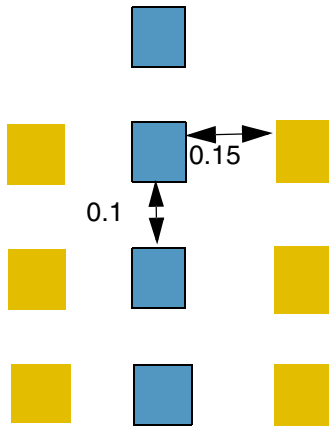
b. Pass: The maximum cluster size is 3 and it has 4 facing edge neighbors. The neighbor at the lower left is not considered because it is only partially facing the edge.

Example 4: *maxViaArrayClusterSize* with *neighborDistanceRange*, *maxFacingEdgeNeighbors*, and *checkEachSideNeighborsSeparately*

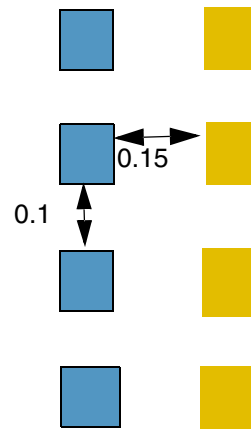
The maximum cluster size is 4, and it can have a maximum of 3 neighbors on each side.

```
spacings (
  ( (maxViaArrayClusterSize "Via1"
    'horizontal
    'mergeSpacing 0
    'clusterSpacingRange "<=0.1"
    'clusterVertical
    'neighborDistanceRange "<=0.15"
    'maxFacingEdgeNeighbors 3
    'checkEachSideNeighborsSeparately
    4
  )
) ;spacings
```

■ Vias in cluster
■ Neighboring vias



a. Pass: The maximum cluster size is 4, and it has 3 neighbors on each side.



b. Fail: The maximum cluster size is 3, but it has more than 3 facing edge neighbors.

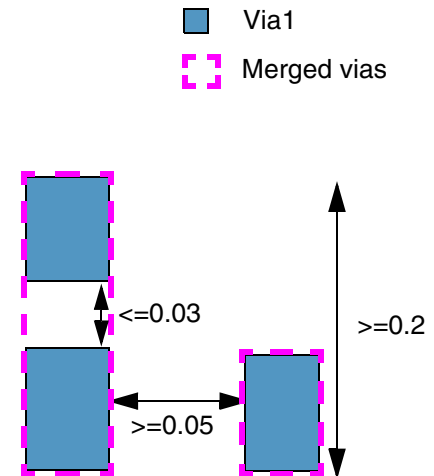
Example 5: *maxViaArrayClusterSize* with *checkLengthOfAnyArray*

Consider the following constraints in an AND group. The via spacing is short-to-short in the first case and long-to-short or long-to-long in the second case.

```

AND Group
1. spacings(
    ( maxViaArrayClusterSize "Via1"
      'vertical
      'mergeSpacing 0.3
      'clusterSpacingRange "<0.4"
      'clusterHorizontal
      'lengthRange "<0.2"
      0.01
    )
) ;spacings
2. spacings(
    ( maxViaArrayClusterSize "Via1"
      'vertical
      'mergeSpacing 0.3
      'clusterSpacingRange "<0.5"
      'clusterHorizontal
      'lengthRange ">=0.2"
      'checkLengthOfAnyArray
      0.01
    )
) ;spacings

```



Pass. The first constraint does not apply because the length of the left merged via is not <0.2 . The second constraint applies because at least one of the merged vias has length ≥ 0.2 and the constraint passes because the spacing is met.

Virtuoso Technology Data Constraints Reference

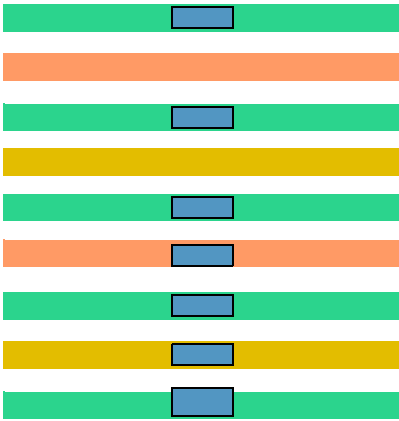
Via Construction Constraints

Example 6: *maxViaArrayClusterSize* with *layer* and *colorMasks*

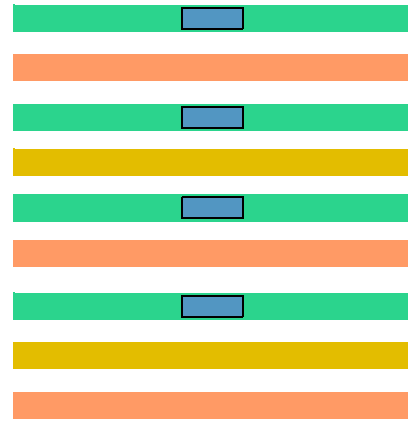
The maximum cluster size is 4. The constraint applies only for cuts on layer Metal2 that are on colorMask 2.

```
spacings (
  ( (maxViaArrayClusterSize "Via1"
    'horizontal
    'mergeSpacing 0.01
    'clusterSpacingRange "<0.02"
    'layer "Metal2" 'colorMasks (2)
    4
  )
) ;spacings
```

■ Via1
■ colorMask 1
■ colorMask 2
■ colorMask 3



a. FAIL: The constraint fails because there are 5 cuts that overlap a mask2 shape of Metal2 in a cluster, but the maximum allowed is 4.



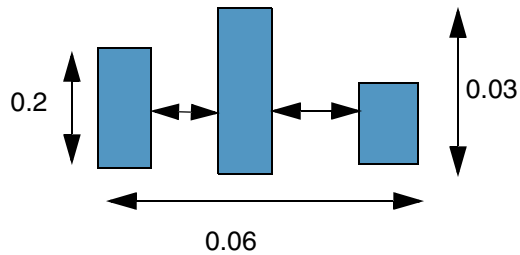
b. PASS: The constraint passes because there are 4 cuts that overlap a mask2 shape of Metal2 in a cluster, which is within the maximum allowed.

Example 7: *maxViaArrayClusterSize* with *clusterLength*

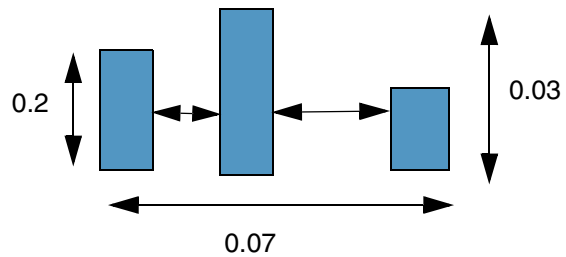
The maximum cluster size is 3, and it can have a maximum of 4 neighbors.

```
spacings (
  (maxViaArrayClusterSize "Via1"
    'vertical
    'mergeSpacing 0.03
    'lengthRange [0.02 0.03]
    'clusterSpacingRange "<=0.01"
    'clusterLength 0.06
    'sameMask 3
  ) ;spacings
```

■ Vias in cluster



a. PASS: The constraint passes because the constraint length is equal to 0.06.



b. FAIL: The constraint fails because the constraint length is greater than 0.06.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

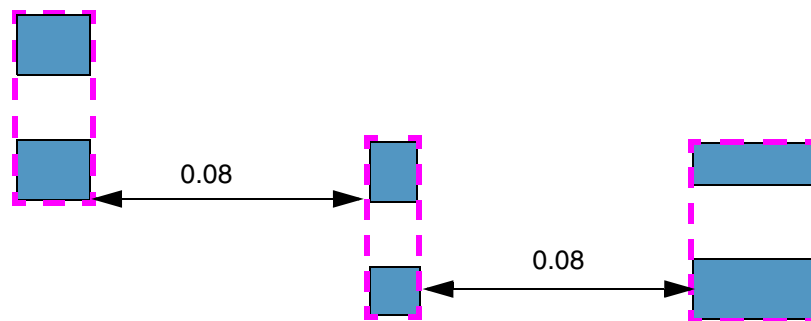
Example 8: *maxViaArrayClusterSize* with *maxNumCutsInArray* and *minClusterSize*

The constraint is met if there are no more than 3 merged vias in the cluster. In both scenarios, the constraint applies because at least one array in the cluster has more than 1 cut but fails because the number of via arrays in the cluster is more than 2.

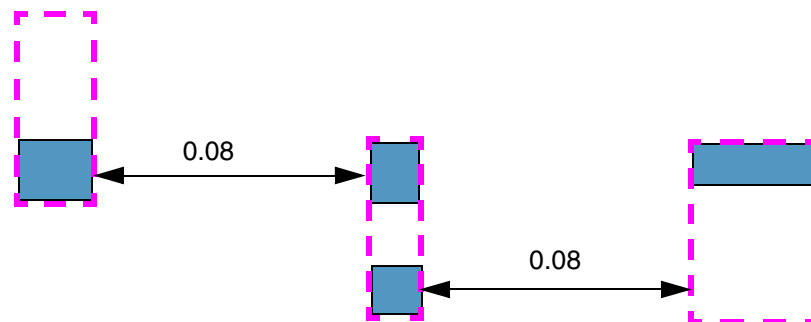
```
spacings(  
  ( (maxViaArrayClusterSize "Vial"  
    'vertical  
    'mergeSpacing 0.03  
    'clusterSpacingRange "<0.088"  
    'sameMask  
    'maxNumCutsInArray 1  
    'minClusterSize 2  
    3  
  )  
) ;spacings
```

■ Cuts
□ Merged vias

Scenario 1



Scenario 2



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

maxViaStack

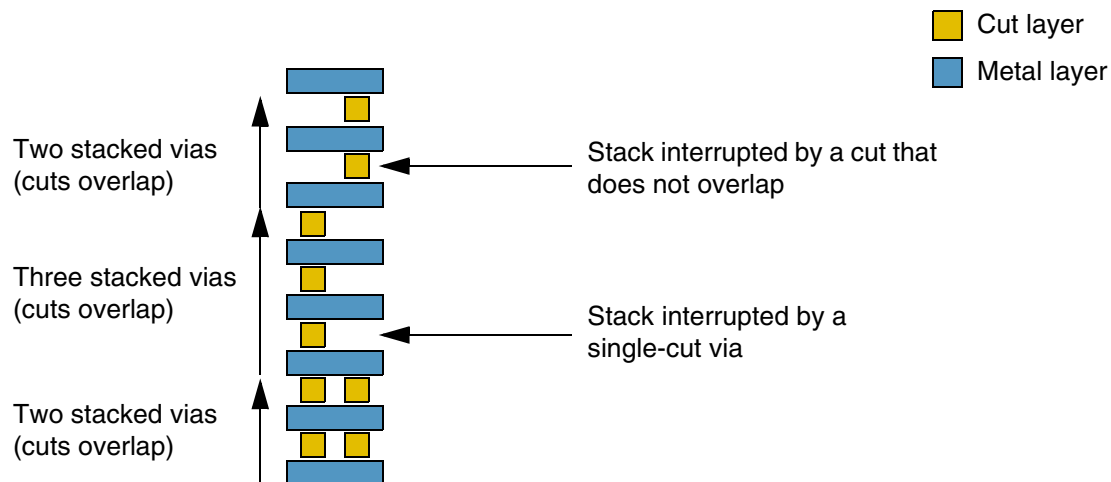
```
spacings (
  ( maxViaStack
    ['lowerLayer tx_lowerLayer]
    ['upperLayer tx_upperLayer]
    ['noSingleCut]
    ['oneSameMetal f_cutWithin]
    ['minMultiCut x_numCut]
    ['lowCutSizeWithin
      | 'withinLayerName l_layerNames 'withinList l_withinValues
      | 'centerToCenterList l_centerToCenterValues]
      | 'layer tx_layerName 'dualWithin (f_belowWithin f_aboveWithin)
    ]
    ['exceptIndividualArea f_individualArea 'exceptSumArea f_sumArea
    | 'exceptMaxArea l_metalLayerNamesAndAreas
    | 'areaRatio l_areaRatio]
    ['exceptPGNet]
    ['exceptPGRouteTopologyList l_PGShapeTypeNames]
    ['exceptCutClassList l_cutClassNames]
    ['excludeEqual x_value]
    ['noViaList l_viaNames]
    ['noCutClassList l_cutClassNames]
    ['sameNetOnly]
    ['exceptExtensionLayerList l_layer
      'exceptExtensionMetalList l_metal
      'exceptExtensionValueList l_value
      'exceptExtensionDirectionList l_direction]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ...t_purposeN)]
    | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ...t_purposeN)]]
    x_stackLimit
  )
) ;spacings
```

Sets the maximum allowed number of continuous stacked vias. A via is considered to be in a stack with another via if a cut in one via partially or completely overlaps a cut in another via. The number of vias in a continuous stack must not exceed the number specified by this constraint.

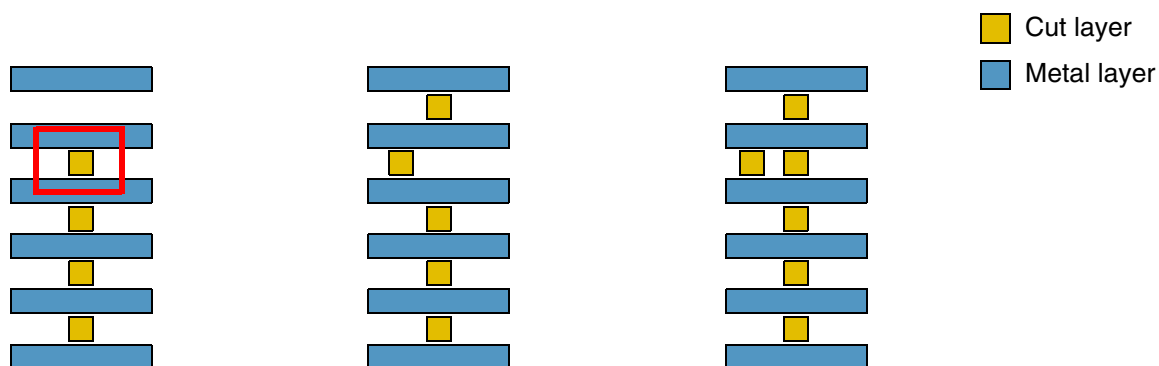
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The figure below illustrates a via stack.



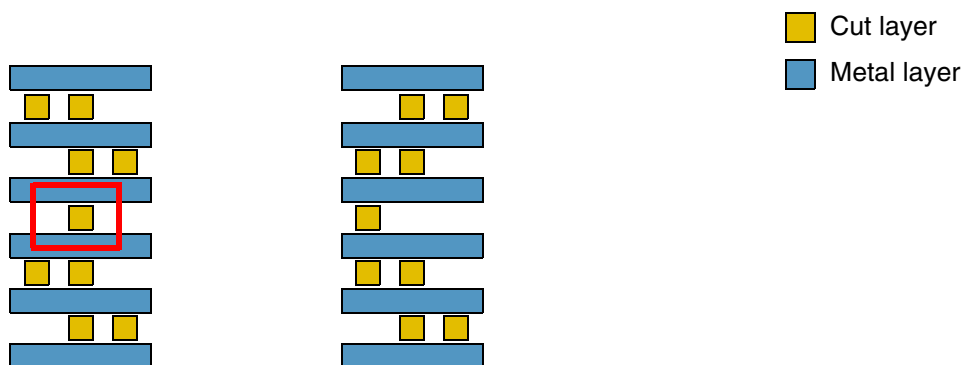
In the first example below, if the via stacking limit is set to 3, the leftmost figure violates the constraint because the number of stacked vias is 4 (>3).



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

In the second example below, if the via stacking limit is set to 3 and 'noSingleCut' is specified, the figure on the left violates the constraint because a single-cut via is not allowed in a stack that is taller than 3.



Values

x_stackLimit

The maximum number of vias that can be stacked.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

`'lowerLayer tx_lowerLayer 'upperLayer tx_upperLayer`

The two together specify a range of layers to which the constraint applies. If this range is not specified, the constraint applies to all layers. The specified layers must all be metal layers.

Type: String (layer name) or Integer (layer number)

`'noSingleCut`

This parameter specifies whether stacked vias can be single-cut or if they must be multiple-cut.

- If this parameter is not specified, single-cut vias can be stacked up to the maximum specified number. Multiple-cut vias interrupt the stack count, effectively resetting the count to zero.
- If this parameter is specified, via stacks taller than the stack count must consist of all multiple-cut vias.

Type: Boolean

`'oneSameMetal f_cutWithin`

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the cuts within the specified value that share at least one common metal are considered as multiple cuts.

`'minMultiCut x_numCut`

(Virtuoso Advanced Node for Layout Standard) This parameter specifies the minimum number of cuts to be considered as a multicut. The default value is 2.

- If `'noSingleCut` is specified, all vias in the stack must have at least `f_numCut` cuts to pass the constraint.
- If `'noSingleCut` is not specified, a via must have at least `f_numCut` cuts to interrupt the stack.

`'lowCutSizeWithin`

It is a stacked via if the upper adjacent layer cut is within the minimum cut size specified on the lower cut layer (as defined in a [cutClasses](#) or [minWidth](#) constraint).

Type: Boolean

`'withinLayerName l_layerNames 'withinList l_withinValues`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Advanced Node for Layout Only) It is a stacked via if the upper adjacent layer cut is within the distance specified by 'withinList of a cut on the lower layer specified with 'withinLayerName.

The number of layer names specified using 'withinLayerName and the number of distance values specified using 'withinList must be the same and must correspond to each other in position.

Type: A list of layer names and a list of floating-point distance values.

`'centerToCenterList l_centerToCenterValues`

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the distances in 'withinList between two cuts on adjacent layers are measured center-to-center.

`'layer tx_layerName 'dualWithin (f_belowWithin f_aboveWithin)`

(Virtuoso Advanced Node for Layout Only) The constraint applies if all cuts below *tx_layerName* are within *f_belowWithin* and all cuts above *tx_layerName* are within *f_aboveWithin*.

Type: A layer name and a pair of floating-point distance values.

`'exceptIndividualArea f_individualArea`

The constraint does not apply if the metal shape containing a stacked via has an area greater than or equal to this value on a layer other than the bottommost and topmost metal layers.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'exceptSumArea f_sumArea`

The constraint does not apply if the union sum of the metal shapes containing a stacked via on layers other than the bottommost and topmost metal layers is greater than or equal to this value.

`'exceptMaxArea l_metalLayerNamesAndAreas`

(Virtuoso Layout Suite EXL and higher tiers) Specifies an exemption when the area on the specified metal layer is less than the specified area on any specified metal layers. The first value for each entry in the list is a metal layer and the second value is the area.

`'areaRatio l_areaRatio`

(Virtuoso Layout Suite EXL and higher tiers) Specifies the area conditions to trigger the rule. Each list entry has three values. The first value specifies a cut layer, the second value specifies the area on that cut layer must be less than or equal to the specified value, and the third value specifies the area of the above or below cut layer or area of the current layer should be greater than the specified value.

`'exceptPGNet`

(Virtuoso Advanced Node for Layout Only) The constraint does not apply to power and ground nets.

Type: Boolean

`'exceptPGRouteTopologyList l_PGShapeTypeNames`

(Virtuoso Layout Suite EXL and higher tiers) Specifies the list of shape types, which the rule is exempted if all the PG cuts belong to the given types.

Valid values: ring, stripe, followpin, iowire, corewire, blockwire, padring, blockring

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'exceptCutClassList *l_cutClassNames*

(Virtuoso Advanced Node for Layout Only) Any vias belonging to the given cut class on the given layer would interrupt a stack as if the vias did not exist for this rule. Each list entry includes a cut layer and two values indicating the dimensions of a cut.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'excludeEqual x_value`

(Virtuoso Layout Suite EXL and higher tiers) The cut should be inside the search window specified by `'lowCutSizeWithin` or `'withinList`. The cut is not considered stacked if it touches the search window.

`'noViaList l_viaNames`

(Virtuoso Advanced Node for Layout Only) This is a list of via names that cannot be used for stacking. The via list should be ordered from the lowest cut layer to the highest cut layer and must be in the range specified by `'lowerLayer` and `'upperLayer`.

You can specify multiple sequences of via names, ordered from the lowest to the highest cut layer, separated using `" , "`. The `'*` wildcard is supported.

Type: A list of via names.

`'sameNetOnly`

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies that the rule applies only on same-net vias.

`'exceptExtensionLayerList l_layer`

(Virtuoso Layout Suite EXL and higher tiers) Exempts all the cuts on the specified cut layers that have an extension on a metal layer specified using the `exceptExtensionMetalList` parameter that is greater than or equal to the values specified by the `exceptExtensionValueList` parameter in the direction specified in the `exceptExtensionDirectionList`.

`'exceptExtensionMetalList l_metal`

(Virtuoso Layout Suite EXL and higher tiers) Exempts all the cuts on the specified cut layers in the `exceptExtensionLayerList` parameter that have an extension on a metal layer specified using the `exceptExtensionMetalList` parameter that is greater than or equal to the value specified using the `exceptExtensionValueList` parameter in the direction specified in the `exceptExtensionDirectionList`.

`'exceptExtensionValueList l_value`

(Virtuoso Layout Suite EXL and higher tiers) Exempts all the cuts on the specified cut layer in the `exceptExtensionLayerList` parameter that have extension on the metal layer specified in the `exceptExtensionMetalList` parameter greater than or equal to the value specified in the direction specified in `exceptExtensionDirectionList`.

`exceptExtensionDirectionList` *l_direction*

(Virtuoso Layout Suite EXL and higher tiers) Exempts all the cuts on the specified cut layer in the `exceptExtensionLayerList` parameter that have an extension on the metal layer specified in the `exceptExtensionMetalList` parameter greater than or equal to the value specified in the `exceptExtensionValueList`.

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

(Virtuoso Layout Suite EXL and higher tiers) Determines if the constraint applies, based on the presence or absence of one or more layers.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*cutLayer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*cutLayer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One Layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

(Virtuoso Layout Suite EXL and higher tiers) The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Layout Suite EXL and higher tiers) The 'outsideLayers' parameter is limited to only the specified purposes.

Type: String (purpose name)

Examples

- Example 1: maxViaStack with withinLayerName and withinList
- Example 2: maxViaStack with noViaList
- Example 3: maxViaStack with noCutClassList
- Example 4: maxViaStack with exceptCutClassList
- Example 5: maxViaStack with dualWithin

Virtuoso Technology Data Constraints Reference

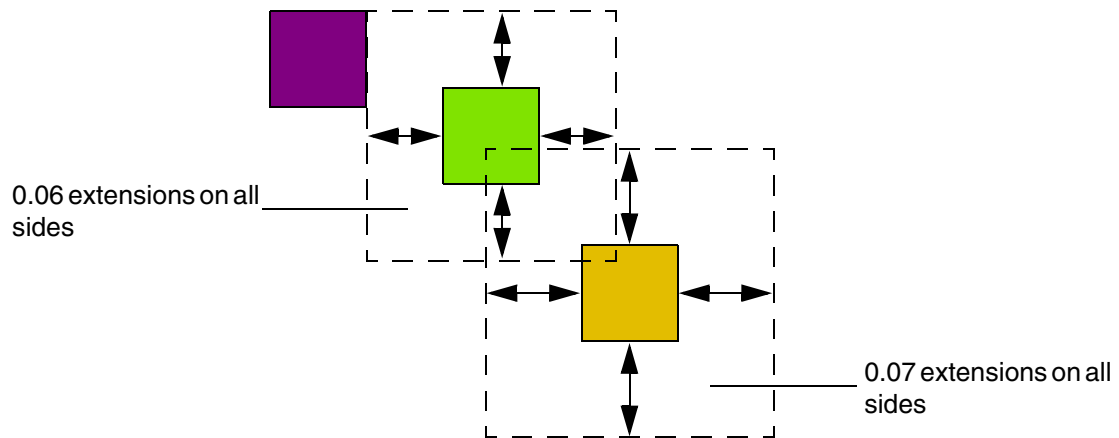
Via Construction Constraints

Example 1: *maxViaStack* with *withinLayerName* and *withinList*

The maximum number of vias in a stack can be three. V2 must be at a distance less than or equal to 0.07 from V1, and V3 must be at a distance less than or equal to 0.06 from V2.

```
spacings(  
  ( maxViaStack  
    'withinLayerName ("V1" "V2") 'withinList (0.07 0.06)  
    3  
  )  
) ;spacings
```

■ V3
■ V2
■ V1



PASS. V1, V2, and V3 are three vias, stacked from V1 to V3. V2 is within 0.07 of V1. V3 touches the V2 *within* search window, and is, therefore, as a distance equal to 0.06 from V2.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 2: maxViaStack with noViaList

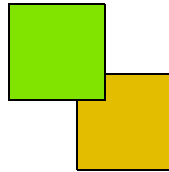
A V1B, V2C via combination is not allowed in a via stack that can contain a maximum of three vias.

The second sequence of via names illustrates how you can use a wildcard to specify via combinations that are not allowed in a via stack.

```
spacings(  
  ( maxViaStack  
    'noViaList ("V1B" "V2C" ", " "V1_*" "*")  
    3  
  )  
) ;spacings
```

 V2C

 V1B



FAIL. Up to 3 vias are allowed in the stack, but a V1B, V2C combination is not allowed.

Example 3: maxViaStack with noCutClassList

The `noCutClassList` parameter here specifies that via stacks containing a cut class of either size 0.06x0.12 or 0.06x0.06 on layer V12 and size 0.06x0.06 on layer V23 are a violation, even if the stack does not exceed the maximum height of 5.

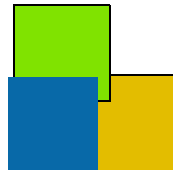
```
spacings(  
  ( maxViaStack 'minMultiCut 3  
    'noCutClassList ("V12 0.06 0.12" "V23 0.06 0.06" ", "  
      "V12 0.06 0.06" "V23 0.06 0.06")  
    5)  
  ) ;spacings
```

Example 4: maxViaStack with exceptCutClassList

A maximum of 2 vias can be stacked in this case, but the stack is interrupted by a 0.1x0.1 cutClass on layer V2.

```
spacings(  
  ( maxViaStack  
    'exceptCutClassList ("V2 0.1 0.1")  
    2)  
  ) ;spacings
```

■ V2
■ V1
■ V3

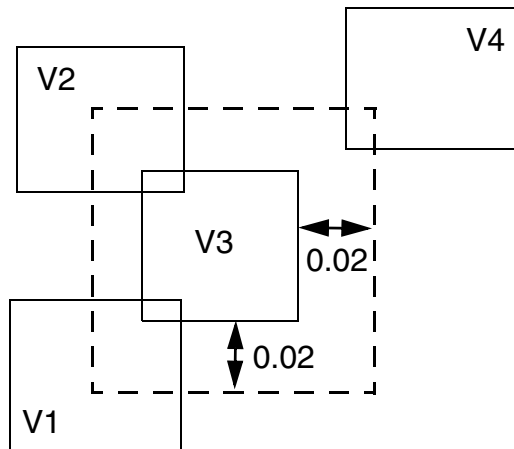


PASS. The stack is interrupted by a V2 via of size 0.1x0.1.
Without `exceptCutClassList`, it would be a violation.

Example 5: maxViaStack with dualWithin

The constraint applies between Metal1 and Metal 5 only when all cuts are as specified by the `dualWithin` values, that is, when all via shapes below V3 are within 0.0 of the V3 shape and all via shapes above V3 are within 0.02 of the V3 shape.

```
spacings (
  ( maxViaStack
    'lowerLayer "Metal1"
    'upperLayer "Metal5"
    'layer "V3" 'dualWithin (0.0 0.02)
    3
  )
) ; spacings
```



FAIL. V1 and V2 overlap with V3, being within 0.0 of V3. V4 is within 0.02 of V3. As a result, the constraint applies and the maximum stack height of 3 is violated.

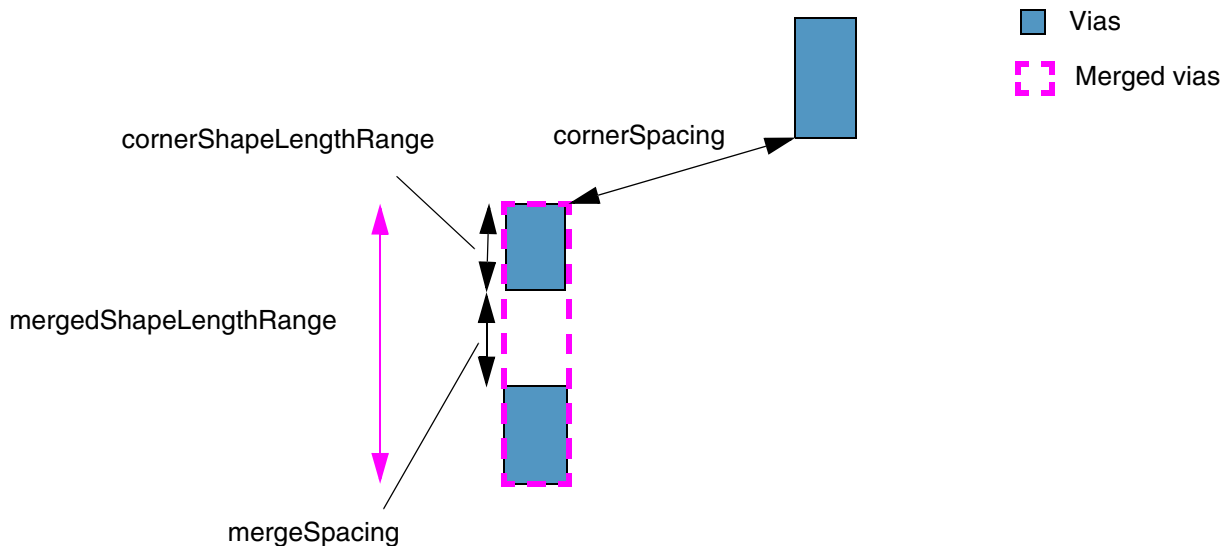
mergedViaCornerToCornerSpacing

```

spacings (
  ( mergedViaCornerToCornerSpacing tx_layer
    ('vertical | 'horizontal
      'mergeSpacing f_mergeSpacing)
    ['enclosingLayer tx_enclosingLayer 'viaEdgeType x_edgeType]
    ['lengthRange g_range1 ['otherLengthRange g_range2]]
    ['tripletCenterViaSpacingX (g_withinX)
      'tripletCenterViaSpacingY (g_withinY)]
    ['tripletCenterViaLarger]
    ['mergedShapeLengthRange g_lengthRange]
    ['oneUnmergedShape]
    ['cornerShapeLengthRange g_cornerLengthRange]
    ['horizontalSpacingRange g_hRange]
    ['verticalSpacingRange g_vRange]
    ['isolatedPairOtherSpacing g_otherSpacing]
    ['maxNeighbors g_maxNeighbors 'neighborSpacing f_neighborSpacing]
    ['sameMask]
    f_cornerSpacing
  )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Defines the corner-to-corner spacing between two merged vias with a negative parallel run length. The exact corner spacing value is checked for without rounding up to mfgGrid value.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_cornerSpacing</i>	The minimum required spacing between merged vias.

Parameters

<code>'vertical 'horizontal</code>	The direction in which shapes should be merged. ■ <code>'vertical</code>: The constraint applies orthogonally in the vertical direction. ■ <code>'horizontal</code>: The constraint applies orthogonally in the horizontal direction. Type: Boolean
<code>'mergeSpacing <i>g_mergeSpacing</i></code>	(Virtuoso Layout Suite EXL and higher tiers) Vias that are spaced at less than or equal to <i>g_mergeSpacing</i> and are exactly aligned are considered for merging into arrays. The alignment is measured orthogonally in the merge direction specified.
<code>'enclosingLayer <i>tx_enclosingLayer</i></code>	(Virtuoso Layout Suite EXL and higher tiers) The enclosing layer for <code>viaEdgeType</code> constraint.
<code>'viaEdgeType <i>x_edgeType</i></code>	(Virtuoso Layout Suite EXL and higher tiers) Vias with cut edges satisfying <code>viaEdgeType</code> constraint are merged before checking corner-to-corner spacing.
<code>'lengthRange <i>g_range1</i></code>	(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if the length of the first shape, which might be merged or unmerged, in the merge direction is in this range.
<code>'otherLengthRange <i>g_range2</i></code>	

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if the length of the other shape, which might be merged or unmerged, is in this range. It depends on *g_range1*.

'tripletCenterViaSpacingX (*g_withinX*)
'tripletCenterViaSpacingY (*g_withinY*)

(Virtuoso Layout Suite EXL and higher tiers) Three cuts at any of these values in the X and Y directions are exempted.

'tripletCenterViaLarger

(Virtuoso Layout Suite EXL and higher tiers) The area of the center via must be greater than or equal to the area of other vias for the diagonal cluster of three vias to be exempted.

'mergedShapeLengthRange *g_lengthRange*

The length of the merged shape must be in this range in the specified direction.

'oneUnmergedShape [*'cornerShapeLengthRange g_cornerLengthRange*]

If *oneUnmergedShape* is specified, one shape must be unmerged during constraint checking. If not specified, shapes to be checked for spacing can be either merged or unmerged.

If *cornerShapeLengthRange* is specified, the corner of the merged shape from which spacing is calculated must have a merged shape in the specified direction.

'horizontalSpacingRange *g_hRange*

The constraint applies only between shapes in this spacing range in the horizontal direction.

'verticalSpacingRange *g_vRange*

The constraint applies only between shapes in this spacing range in the vertical direction.

'isolatedPairOtherSpacing *g_otherSpacing*

The constraint applies only if other shapes have at least the specified spacing from both the shapes or from all the three shapes in a cluster in any direction.

'maxNeighbors *g_maxNeighbors*

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Layout Suite EXL and higher tiers) This constraint is violated if there are more than the specified number of neighbour vias within the spacing specified by 'neighborSpacing.

'neighborSpacing *f_neighborSpacing*

(Virtuoso Layout Suite EXL and higher tiers) This constraint is violated if there are more than 'maxNeighbors of vias within this spacing.

'sameMask

If specified, the constraint applies only between shapes on the same mask.

Examples

- Example 1: mergedViaCornerToCornerSpacing with isolatedPairOtherSpacing
- Example 2: mergedViaCornerToCornerSpacing with tripletCenterViaSpacing
- Example 3: mergedViaCornerToCornerSpacing with lengthRange
- Example 4: mergedViaCornerToCornerSpacing with maxNeighbors and neighborSpacing

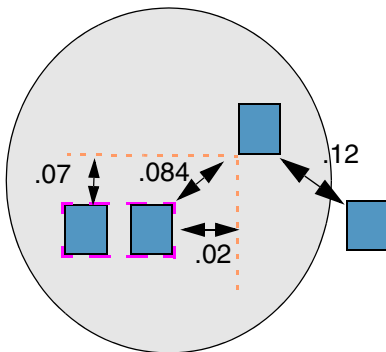
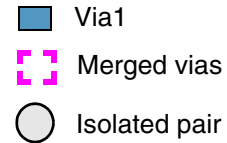
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

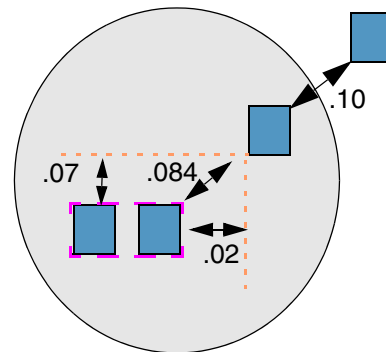
Example 1: mergedViaCornerToCornerSpacing with isolatedPairOtherSpacing

Consider the following example and scenarios.

```
spacings(  
  ( mergedViaCornerToCornerSpacing "Via1"  
    'horizontal  
    'mergeSpacing .02  
    'mergedShapeLengthRange ">=.09"  
    'oneUnmergedShape  
      'cornerShapeLengthRange "<=.03"  
    'horizontalSpacingRange "<=0.02"  
    'verticalSpacingRange ">=0.065"  
    'isolatedPairOtherSpacing .11  
    'sameMask  
    .083  
  )  
) ;spacings
```



Pass. Spacing in isolated pair $\geq .083$.



The constraint does not apply because the other shape is at $< .11$ from the isolated pair; so, it is not an isolated pair.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 2: mergedViaCornerToCornerSpacing with tripletCenterViaSpacing

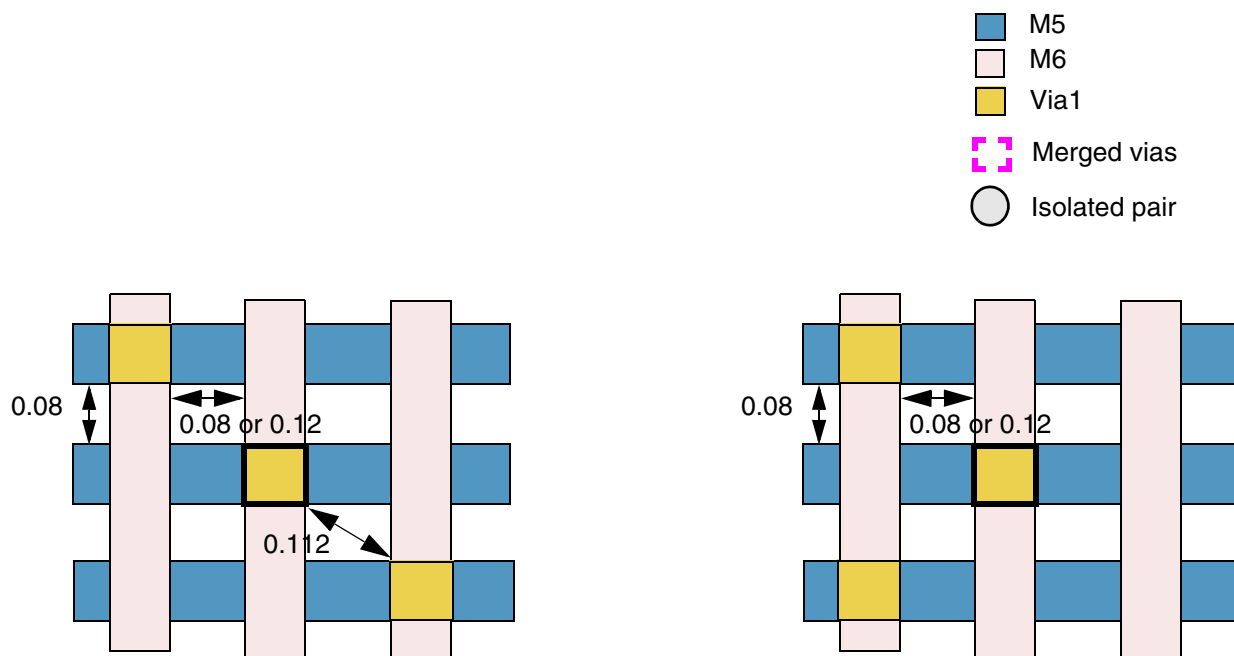
Consider the following example, which shows an OR group with three constraints. The first constraint is a regular corner-to-corner constraint. The second also includes `isolatedOtherPairSpacing`. The third further includes triplet via spacing parameters.

OR group

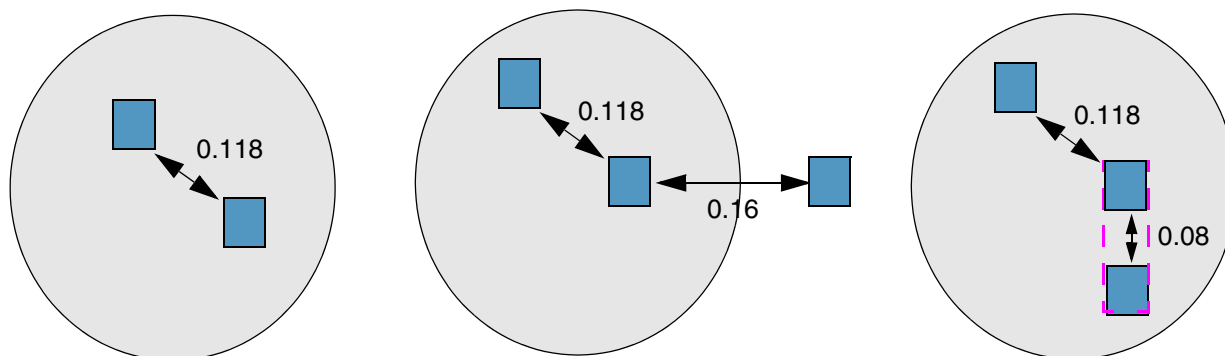
```
( mergedViaCornerToCornerSpacing "Via1"
  'vertical
  'mergeSpacing 0.216
  0.16
  'ref "mergeViaOr1"
)
( mergedViaCornerToCornerSpacing "Via1"
  'vertical
  'mergeSpacing 0.216
  'isolatedOtherPairSpacing 0.16
  0.112
  'ref "mergeViaOr2"
)
( mergedViaCornerToCornerSpacing "Via1"
  'vertical
  'mergeSpacing 0.216
  'isolatedOtherPairSpacing 0.16
  'tripletCenterViaSpacingX (0.08 0.12)
  'tripletCenterViaSpacingY (0.08)
  'tripletCenterViaLarger
  0.112
  'ref "mergeViaOr3"
)
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints



The third constraint applies and passes in these cases because they form valid diagonal clusters.

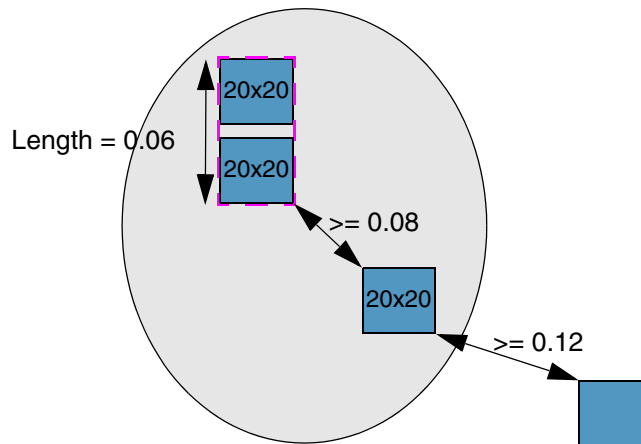


The second constraint applies and passes because these have an isolated pair.

Example 3: mergedViaCornerToCornerSpacing with lengthRange

Consider the following example and scenarios.

```
spacings(
  ( mergedViaCornerToCornerSpacing "Via1"
    'vertical
    'mergeSpacing 0.02
    'isolatedOtherPairSpacing 0.12
    'lengthRange 0.06
    'otherLengthRange 0.02
    0.08
  )
) ;spacings
```



Pass. The constraint applies because the length of one group is 0.06 and the length of the other group is 0.02. The spacing of ≥ 0.08 is met.

Example 4: mergedViaCornerToCornerSpacing with maxNeighbors and neighborSpacing

Consider the following example and scenarios.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

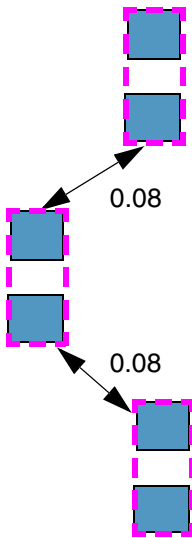
OR group

```
(mergedViaCornerToCornerSpacing "Via1" 'vertical  
'mergeSpacing 0.02 0.10)
```

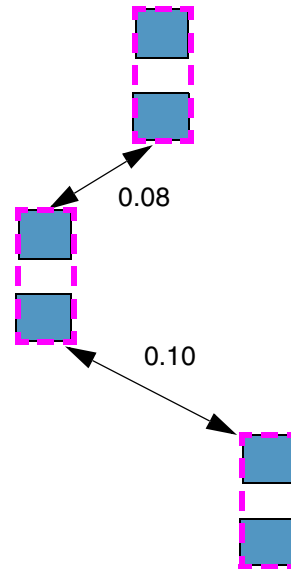
```
(mergedViaCornerToCornerSpacing "Via1" 'vertical  
'mergeSpacing 0.02 'maxNeighbors 1 'neighborSpacing 0.10 0.08)
```

■ Vias

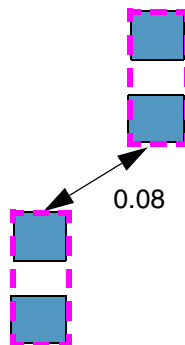
□ Merged vias



FAIL: There are 2 neighbors within 0.1 but only one neighbor is allowed.



PASS: There is only one neighbor within 0.1



PASS: There is only one neighbor within 0.1

minCutClassSpacing (One layer)

```
spacingTables(
  ( minCutClassSpacing tx_cutLayer
    (( "cutClass" (g_index ...) nil "cutClass" (g_index ...) nil )
      ['sameNet | 'sameMetal | 'sameVia]
      ['sameMask] ['bothNegativePrls]
      ['exactAligned ((f_width | t_name f_spacing) ...)]
      ['horizontal | 'vertical]
      ['paraOverlap f_overlap
        ['horizontalOverlap | 'verticalOverlap]
      ]
      ['nonCutClassEdgeSpacing (f_length1 f_space1 f_length2 f_space2 ...)]
      ['cutClassProfile
        (( "cutClass" (g_index ...) nil "cutClass" (g_index ...) nil )
          (g_cutClassProfileTable)
        )
      ]
      ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
        |
        ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
          ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      ]
      [f_default]
    )
    (g_table) ['manhattan]
  )
) ;spacingTables
```

Specifies the spacing between cut shapes of the same or of different classes on a layer. The spacing requirement depends on the size of the cut shape and on the edge—long or short—from which the spacing is measured.

Optional parameters determine whether spacing is measured center-to-center or edge-to-edge, whether the constraint applies to a certain connectivity type, and whether the constraint applies only if the cut shapes have a parallel run length greater than or equal to the specified value.

The optional `'bothNegativePrls` parameter determines how parallel run lengths less than zero are handled.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

tx_layer The cut layer on which the constraint is applied.
Type: String (layer and purpose names) or Integer (layer number)

"cutClass" (*g_index* ...) nil "cutClass" (*g_index* ...) nil

This identifies the index for *table*.

The format of *g_index* is as follows:

```
f_width | (f_width f_length) | t_className |  
((f_width f_length) 'shortEdge | 'longEdge) |  
(t_className 'shortEdge | 'longEdge)
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

g_table

The spacing between the long edges (sides) and/or short edges (ends) of cuts, of the same class or of different classes.

The *table* row has the following format:

```
((f_edgeLength | t_className 'shortEdge |  
'longEdge) (f_edgeLength | t_className  
'shortEdge | 'longEdge)) f_spacing |  
(f_spacing ['centerToCenter | 'centerAndEdge |  
'centerAndEdgeNoPRL])
```

where,

- *f_edgeLength* is the width or length of the cut class.
- *t_className* is the name of the cut class and must correspond to a cut class defined in a [cutClasses](#) constraint.
- 'shortEdge and 'longEdge indicate the edge of the cut shape from which spacing is measured.
- *f_spacing* is the minimum spacing required between cut shapes.
- 'centerToCenter indicates that spacing is measured from the center of one cut shape to the center of another cut shape.
- 'centerAndEdge specifies that the maximum of all applicable spacing values must be measured center-to-center and the minimum of all applicable spacing values must be measured edge-to-edge.
- 'centerAndEdgeNoPRL specifies that the spacing between two cut shapes must be the greater of the following two values:
 - The maximum of all applicable values, which is measured center-to-center.
 - The minimum of all applicable values, which is measured edge-to-edge.

Type: A 2-D table specifying floating-point width or length and spacing values

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

'sameNet | 'sameMetal | 'sameVia

The connectivity type. If connectivity type is not specified, the constraint applies to any connectivity type, including any two via cuts on the same cut layer with no common metal or net.

- 'sameNet: The constraint applies only if the via cuts are on the same net.
- 'sameMetal: The constraint applies only if the via cuts are on a contiguous same-metal shape.
- 'sameVia: The constraint applies only if the via cuts are overlapped by a single metal shape from above and by another single metal shape from below.

Type: Boolean

'centerToCenter

The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge.

Type: Boolean

'sameMask

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to via cuts on the same mask.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

<code>'bothNegativePrls</code>	(Virtuoso Advanced Node for Layout Standard) If the parallel run length between the cut shapes is less than zero, this parameter allows both horizontal and vertical offsets between the cut shapes to be used as parallel run lengths. 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 <code>minCutClassSpacing</code> constraint defined on a particular layer, it must be specified for all <code>minCutClassSpacing</code> constraints defined on that layer. Type: Boolean
<code>'exactAligned ((f_width t_name f_spacing) ...)</code>	The spacing between square cut shapes with the specified width or between cut shapes that belong to the specified cut class must be greater than or equal to this value if the cut shapes are exactly aligned, vertically or horizontally.
<code>'horizontal 'vertical</code>	The direction in which the constraint applies. If direction is not specified, the constraint applies in any direction. Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'paraOverlap *f_overlap*

The constraint applies only if the parallel run length (prl) between the cut shapes is greater than or equal to this value. Positive and negative values are allowed.

If multiple `minCutClassSpacing` constraints are defined in an AND constraint group, the constraint is applied by evaluating the *overlap* values, specified in the ascending order, to locate the first *overlap* value for which the parallel run length is greater than or equal to this value. As a result, only one constraint in an AND constraint group applies. For example, in an AND constraint group with parallel run length values specified as -1.0, -0.05, 0, 0.05, 1, the constraint value that applies is determined as follows:

<i>overlap</i> (prl)	Use constraint value for <i>overlap</i>
<-1.0	The constraint does not apply
>=-1.0 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

'horizontalOverlap | 'verticalOverlap

(Virtuoso Advanced Node for Layout Only) The direction in which parallel run length is measured. By default, parallel run length is measured in any direction.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
'cutClassProfile (( "cutClass" (g_index ...) nil "cutClass" (g_index ...) nil ) (g_cutClassProfileTable))
```

The cut shapes with the specified dimensions or class are expanded by a value equal to *extension* before spacing is measured.

The table has the following format:

```
((f_edgeLength | t_className  
  'shortEdge | 'longEdge  
)  
(f_edgeLength | t_className  
  'shortEdge | 'longEdge  
)  
) f_extension
```

This value is applied only if the parallel run length between the cut shapes is less than or equal to zero.

Type: A 2-D table specifying floating-point width or length values and the value by which the cut shapes need to be expanded.

```
'nonCutClassEdgeSpacing (f_length1 f_space1)
```

(ICADVM20.1 Only–95512) The spacing between a cut shape and a non-cut-class shape, such as a blockage, of length *length1* must be greater than or equal to *space1*.

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*cutLayer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*cutLayer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One Layer\)](#).

Type: String (layer name) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

`f_default`

The spacing value to be used when no table entry applies.

`'manhattan`

The constraint uses Manhattan distance, which allows a larger spacing at the corners.

By default, the constraint uses Euclidian measurement.

Type: Boolean

Examples

- [Example 1: minCutClassSpacing with paraOverlap](#)
- [Example 2: minCutClassSpacing with paraOverlap and cutClassProfile](#)
- [Example 3: minCutClassSpacing without bothNegativePrIs](#)
- [Example 4: minCutClassSpacing with bothNegativePrIs](#)

Example 1: minCutClassSpacing with paraOverlap

Three cut classes are defined on Via1: VA, VB, and VC. The spacing required between cut shapes of different cut classes is as follows:

- Center-to-center spacing between two VA via cuts is 0.20.
- Edge-to-edge spacing between the short edge of a VB via cut and a VA or VC via cut if parallel run length is greater than zero is 0.30.
- Edge-to-edge spacing between the short edge of a VB via cut and an edge of another VB via cut if parallel run length is greater than zero is 0.40.
- Center-to-center spacing between two VC via cuts is 0.50.

For the rest of the combinations, a default spacing of 0.15 applies.

```
layerRules(  
  cutClasses(  
    (Via1  
      (VA 0.3)  
      (VB (0.4 0.6))  
      (VC 0.7)  
    )  
  ) ;cutClasses  
) ;layerRules
```

Virtuoso Technology Data Constraints Reference

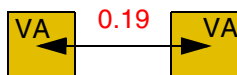
Via Construction Constraints

```
spacingTables(  
  ( minCutClassSpacing "Via1"  
    (( "cutClass" ("VA" ("VB" 'shortEdge) ("VB" 'longEdge) "VC") nil  
      "cutClass" nil nil )  
      'paraOverlap 0.001  
      0.15  
    )  
    (  
      ((("VA" 'shortEdge) ("VA" 'shortEdge)) (0.20 'centerToCenter)  
        ((("VA" 'shortEdge) ("VA" 'longEdge )) (0.20 'centerToCenter)  
        ((("VA" 'shortEdge) ("VB" 'shortEdge)) 0.30  
        ((("VA" 'shortEdge) ("VB" 'longEdge )) 0.15  
        ((("VA" 'shortEdge) ("VC" 'shortEdge)) 0.15  
        ((("VA" 'shortEdge) ("VC" 'longEdge )) 0.15  
        ((("VA" 'longEdge ) ("VA" 'shortEdge)) (0.20 'centerToCenter)  
        ((("VA" 'longEdge ) ("VA" 'longEdge )) (0.20 'centerToCenter)  
        ((("VA" 'longEdge ) ("VB" 'shortEdge)) 0.30  
        ((("VA" 'longEdge ) ("VB" 'longEdge )) 0.15  
        ((("VA" 'longEdge ) ("VC" 'shortEdge)) 0.15  
        ((("VA" 'longEdge ) ("VC" 'longEdge )) 0.15  
        ((("VB" 'shortEdge) ("VA" 'shortEdge)) 0.30  
        ((("VB" 'shortEdge) ("VA" 'longEdge )) 0.30  
        ((("VB" 'shortEdge) ("VB" 'shortEdge)) 0.40  
        ((("VB" 'shortEdge) ("VB" 'longEdge )) 0.40  
        ((("VB" 'shortEdge) ("VC" 'shortEdge)) 0.30  
        ((("VB" 'shortEdge) ("VC" 'longEdge )) 0.30  
        ((("VB" 'longEdge ) ("VA" 'shortEdge)) 0.15  
        ((("VB" 'longEdge ) ("VA" 'longEdge )) 0.15  
        ((("VB" 'longEdge ) ("VB" 'shortEdge)) 0.40  
        ((("VB" 'longEdge ) ("VB" 'longEdge )) 0.15  
        ((("VB" 'longEdge ) ("VC" 'shortEdge)) 0.15  
        ((("VB" 'longEdge ) ("VC" 'longEdge )) 0.15  
        ((("VC" 'shortEdge) ("VA" 'shortEdge)) 0.15  
        ((("VC" 'shortEdge) ("VA" 'longEdge )) 0.15  
        ((("VC" 'shortEdge) ("VB" 'shortEdge)) 0.30  
        ((("VC" 'shortEdge) ("VB" 'longEdge )) 0.15  
        ((("VC" 'shortEdge) ("VC" 'shortEdge)) (0.50 'centerToCenter)  
        ((("VC" 'shortEdge) ("VC" 'longEdge )) (0.50 'centerToCenter)  
        ((("VC" 'longEdge ) ("VA" 'shortEdge)) 0.15  
        ((("VC" 'longEdge ) ("VA" 'longEdge )) 0.15  
        ((("VC" 'longEdge ) ("VB" 'shortEdge)) 0.30  
        ((("VC" 'longEdge ) ("VB" 'longEdge )) 0.15  
        ((("VC" 'longEdge ) ("VC" 'shortEdge)) (0.50 'centerToCenter)  
        ((("VC" 'longEdge ) ("VC" 'longEdge )) (0.50 'centerToCenter)  
      )  
    )  
  ) ;spacingTables
```

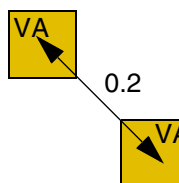
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

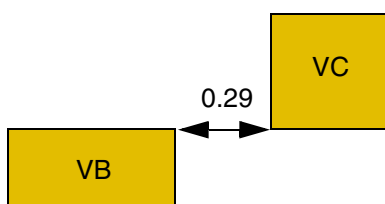
■ Via1



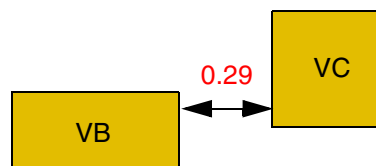
a) FAIL. The center-to-center spacing between the two shapes is 0.19 (< 0.2).



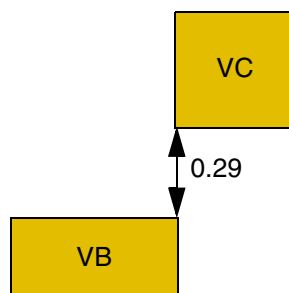
b) PASS. The center-to-center spacing between the two shapes is 0.2



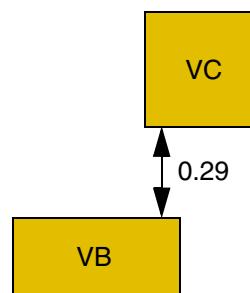
c) PASS. The parallel run length between a VB short edge and a VC edge is zero. Therefore, a default minimum spacing of 0.15 applies ($0.29 > 0.15$).



d) FAIL. The spacing between a VB short edge and a VC edge, when parallel run length is greater than zero, must be at least 0.30 ($0.29 < 0.30$).



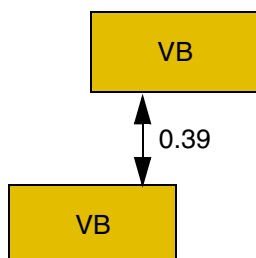
e) PASS. The parallel run length between a VB short edge and a VC edge is zero. Therefore, a default minimum spacing of 0.15 applies ($0.29 > 0.15$).



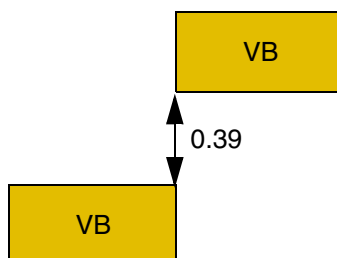
f) PASS. The spacing between a VB long edge and a VC edge, when parallel run length is greater than 0, must be at least 0.15 ($0.29 > 0.15$).

Virtuoso Technology Data Constraints Reference

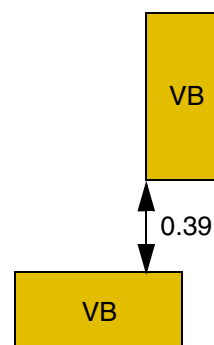
Via Construction Constraints



g) FAIL. The spacing between two VB long edges must be at least 0.40 ($0.39 < 0.40$).



h) PASS. The parallel run length between a VB long edge and a VB short edge is zero. Therefore, a default spacing of 0.15 applies ($0.39 > 0.15$).



i) PASS. The spacing between a VB long edge and a VB short edge, when parallel run length is greater than zero, must be at least 0.15 ($0.39 > 0.15$).

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 2: minCutClassSpacing with paraOverlap and cutClassProfile

Three cut classes are defined on Via1: VA, VB, and VC. The minCutClassSpacing constraint is defined as follows for shapes of these cut classes:

```
spacingTables (
  ( minCutClassSpacing "Via1"
    (( "cutClass" ("VA" ("VB" 'shortEdge) ("VB" 'longEdge) "VC") nil
      "cutClass" nil nil )
    'paraOverlap 0
    'cutClassProfile
      (( "cutClass" (("VB" 'shortEdge) ("VB" 'longEdge)) nil
        "cutClass" ("VA" ("VB" 'shortEdge) ("VB" 'longEdge) "VC") nil )
        (
          (( "VB" 'shortEdge) ("VA" 'shortEdge)) 0.02
          (( "VB" 'shortEdge) ("VA" 'longEdge )) 0.02
          (( "VB" 'shortEdge) ("VB" 'shortEdge)) 0.0
          (( "VB" 'shortEdge) ("VB" 'longEdge )) 0.0
          (( "VB" 'shortEdge) ("VC" 'shortEdge)) 0.02
          (( "VB" 'shortEdge) ("VC" 'longEdge )) 0.02

          (( "VB" 'longEdge ) ("VA" 'shortEdge)) 0.0
          (( "VB" 'longEdge ) ("VA" 'longEdge )) 0.0
          (( "VB" 'longEdge ) ("VB" 'shortEdge)) 0.0
          (( "VB" 'longEdge ) ("VB" 'longEdge )) -0.02
          (( "VB" 'longEdge ) ("VC" 'shortEdge)) 0.0
          (( "VB" 'longEdge ) ("VC" 'longEdge )) 0.0
        )
      )
    0.15 )
  (
    (( "VA" 'shortEdge) ("VA" 'shortEdge)) (0.20 'centerToCenter)
    (( "VA" 'shortEdge) ("VA" 'longEdge )) (0.20 'centerToCenter)
    (( "VA" 'shortEdge) ("VB" 'shortEdge)) 0.30
    (( "VA" 'shortEdge) ("VB" 'longEdge )) 0.15
    (( "VA" 'shortEdge) ("VC" 'shortEdge)) 0.15
    (( "VA" 'shortEdge) ("VC" 'longEdge )) 0.15

    (( "VA" 'longEdge ) ("VA" 'shortEdge)) (0.20 'centerToCenter)
    (( "VA" 'longEdge ) ("VA" 'longEdge )) (0.20 'centerToCenter)
    (( "VA" 'longEdge ) ("VB" 'shortEdge)) 0.30
    (( "VA" 'longEdge ) ("VB" 'longEdge )) 0.15
    (( "VA" 'longEdge ) ("VC" 'shortEdge)) 0.15
    (( "VA" 'longEdge ) ("VC" 'longEdge )) 0.15

    (( "VB" 'shortEdge) ("VA" 'shortEdge)) 0.30
    (( "VB" 'shortEdge) ("VA" 'longEdge )) 0.30
    (( "VB" 'shortEdge) ("VB" 'shortEdge)) 0.40
    (( "VB" 'shortEdge) ("VB" 'longEdge )) 0.40
    (( "VB" 'shortEdge) ("VC" 'shortEdge)) 0.30
    (( "VB" 'shortEdge) ("VC" 'longEdge )) 0.30

    (( "VB" 'longEdge ) ("VA" 'shortEdge)) 0.15
    (( "VB" 'longEdge ) ("VA" 'longEdge )) 0.15
    (( "VB" 'longEdge ) ("VB" 'shortEdge)) 0.40
    (( "VB" 'longEdge ) ("VB" 'longEdge )) 0.15
    (( "VB" 'longEdge ) ("VC" 'shortEdge)) 0.15
    (( "VB" 'longEdge ) ("VC" 'longEdge )) 0.15

    (( "VC" 'shortEdge) ("VA" 'shortEdge)) 0.15
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```

    (("VC" 'shortEdge) ("VA" 'longEdge )) 0.15
    (("VC" 'shortEdge) ("VB" 'shortEdge)) 0.30
    (("VC" 'shortEdge) ("VB" 'longEdge )) 0.15
    (("VC" 'shortEdge) ("VC" 'shortEdge)) (0.50 'centerToCenter)
    (("VC" 'shortEdge) ("VC" 'longEdge )) (0.50 'centerToCenter)

    (("VC" 'longEdge ) ("VA" 'shortEdge)) 0.15
    (("VC" 'longEdge ) ("VA" 'longEdge )) 0.15
    (("VC" 'longEdge ) ("VB" 'shortEdge)) 0.30
    (("VC" 'longEdge ) ("VB" 'longEdge )) 0.15
    (("VC" 'longEdge ) ("VC" 'shortEdge)) (0.50 'centerToCenter)
    (("VC" 'longEdge ) ("VC" 'longEdge )) (0.50 'centerToCenter)
  )
) ;spacingTables

```

The 'cutClassProfile parameter uses its own 2-D table, as shown below:

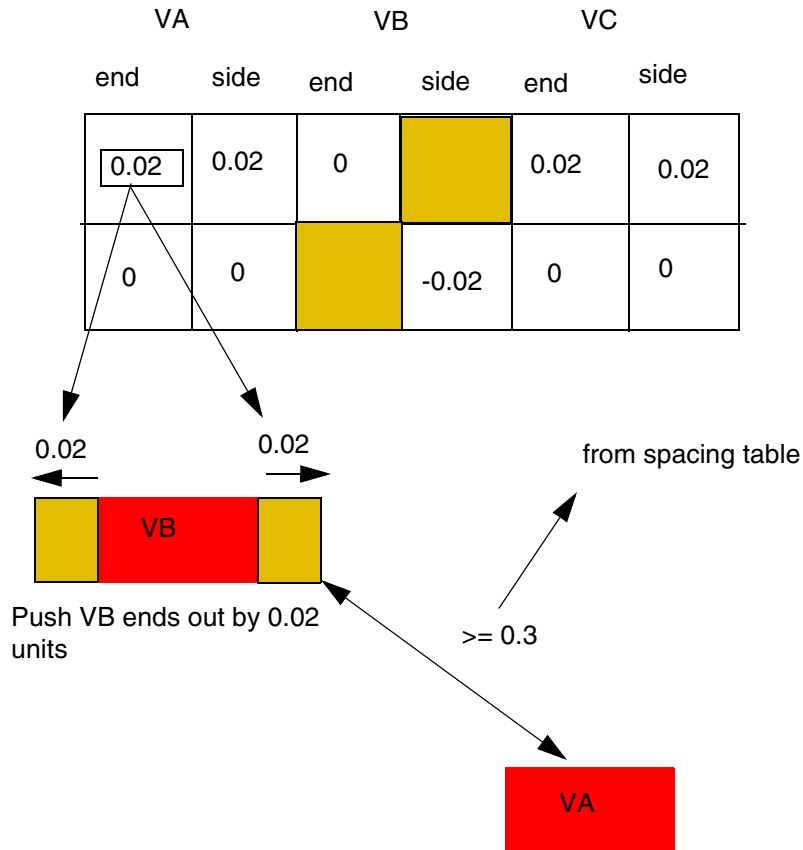
			VA		VB		VC	
			SE	LE	SE	LE	SE	LE
			0.3	0.3	0.4	0.6	0.7	0.7
VB	SE	0.4	0.02	0.02	0.0	0.0	0.02	0.02
	LE	0.6	0.0	0.0	0.0	-0.02	0.0	0.0

For square cut shapes, VA and VC, the LE ('longEdge) and SE ('shortEdge) entries must be equal, and for rectangular cuts, only LE/LE and SE/SE entries are valid. Therefore, the two

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

entries in italics are ignored. Additionally, only the dimensions of rectangular cut shapes are used to index rows.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 3: minCutClassSpacing without bothNegativePrIs

The spacing defined between two cut shapes of type VA (0.2x0.1) and VB (0.2x0.4) for parallel run length (PRL) -0.1 and -0.2 is as follows:

PRL = -0.1

	0.1	0.2	0.2	0.4
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

PRL = -0.2

	0.1	0.2	0.2	0.4
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

Virtuoso Technology Data Constraints Reference

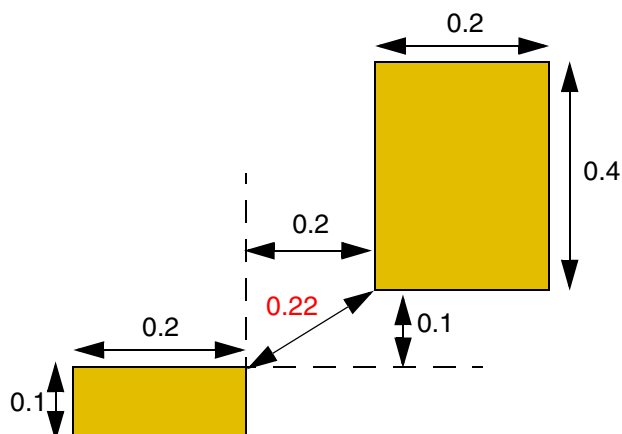
Via Construction Constraints

```
spacingTables (
  ( minCutClassSpacing "Via1"
    (( "cutClass" (("VA" 'shortEdge) ("VA" 'longEdge)
                  ("VB" 'shortEdge) ("VB" 'longEdge)) nil
      "cutClass" nil nil )
    'paraOverlap -0.1
  )
  (
    (("VA" 'shortEdge) ("VA" 'shortEdge)) 0.20
    (("VA" 'shortEdge) ("VA" 'longEdge ) ) 0.20
    (("VA" 'shortEdge) ("VB" 'shortEdge)) 0.20
    (("VA" 'shortEdge) ("VB" 'longEdge ) ) 0.22
    (("VA" 'longEdge ) ("VA" 'shortEdge)) 0.20
    (("VA" 'longEdge ) ("VA" 'longEdge ) ) 0.20
    (("VA" 'longEdge ) ("VB" 'shortEdge)) 0.21
    (("VA" 'longEdge ) ("VB" 'longEdge ) ) 0.23
    (("VB" 'shortEdge) ("VA" 'shortEdge)) 0.20
    (("VB" 'shortEdge) ("VA" 'longEdge ) ) 0.21
    (("VB" 'shortEdge) ("VB" 'shortEdge)) 0.30
    (("VB" 'shortEdge) ("VB" 'longEdge ) ) 0.30
    (("VB" 'longEdge ) ("VA" 'shortEdge)) 0.21
    (("VB" 'longEdge ) ("VA" 'longEdge ) ) 0.23
    (("VB" 'longEdge ) ("VB" 'shortEdge)) 0.30
    (("VB" 'longEdge ) ("VB" 'longEdge ) ) 0.30
  )
)
( minCutClassSpacing "Via1"
  (( "cutClass" (("VA" 'shortEdge) ("VA" 'longEdge)
                  ("VB" 'shortEdge) ("VB" 'longEdge)) nil
      "cutClass" nil nil )
  'paraOverlap -0.2
)
(
  (("VA" 'shortEdge) ("VA" 'shortEdge)) 0.20
  (("VA" 'shortEdge) ("VA" 'longEdge ) ) 0.20
  (("VA" 'shortEdge) ("VB" 'shortEdge)) 0.20
  (("VA" 'shortEdge) ("VB" 'longEdge ) ) 0.22
  (("VA" 'longEdge ) ("VA" 'shortEdge)) 0.20
  (("VA" 'longEdge ) ("VA" 'longEdge ) ) 0.20
  (("VA" 'longEdge ) ("VB" 'shortEdge)) 0.21
  (("VA" 'longEdge ) ("VB" 'longEdge ) ) 0.24
  (("VB" 'shortEdge) ("VA" 'shortEdge)) 0.20
  (("VB" 'shortEdge) ("VA" 'longEdge ) ) 0.21
  (("VB" 'shortEdge) ("VB" 'shortEdge)) 0.30
  (("VB" 'shortEdge) ("VB" 'longEdge ) ) 0.30
  (("VB" 'longEdge ) ("VA" 'shortEdge)) 0.21
  (("VB" 'longEdge ) ("VA" 'longEdge ) ) 0.23
  (("VB" 'longEdge ) ("VB" 'shortEdge)) 0.30
  (("VB" 'longEdge ) ("VB" 'longEdge ) ) 0.30
)
) ;spacingTables
```

■ Via1

Virtuoso Technology Data Constraints Reference

Via Construction Constraints



FAIL. Because 'bothNegativePrIs is not specified, the smaller of the two offsets, 0.1, is used. PRL = -1*offset; therefore, the constraint table with 'paraOverlap value of -0.1 is looked up to determine spacing required between the two shapes. This provides four values—0.2, 0.22, 0.21, 0.23 (indicated in blue in the table above)—of which 0.23 is the largest. The Euclidean spacing between the shapes is 0.22, which is less than 0.23.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 4: minCutClassSpacing with bothNegativePrIs

The spacing defined between two cut shapes of type VA (0.2x0.1) and VB (0.2x0.4) for parallel run length (PRL) -0.1 and -0.2 is as follows:

PRL = -0.1

	0.1	0.2	0.2	0.4
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

PRL = -0.2

	0.1	0.2	0.2	0.4
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

Virtuoso Technology Data Constraints Reference

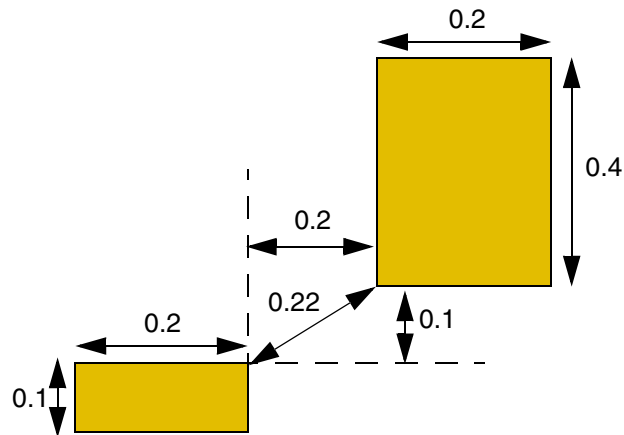
Via Construction Constraints

```
spacingTables (
  ( minCutClassSpacing "Via1"
    (( "cutClass" (("VA" 'shortEdge) ("VA" 'longEdge)
                  ("VB" 'shortEdge) ("VB" 'longEdge)) nil
      "cutClass" nil nil )
    'paraOverlap -0.1 'bothNegativePrls
  )
  (
    (("VA" 'shortEdge) ("VA" 'shortEdge)) 0.20
    (("VA" 'shortEdge) ("VA" 'longEdge ) ) 0.20
    (("VA" 'shortEdge) ("VB" 'shortEdge)) 0.20
    (("VA" 'shortEdge) ("VB" 'longEdge ) ) 0.22
    (("VA" 'longEdge ) ("VA" 'shortEdge)) 0.20
    (("VA" 'longEdge ) ("VA" 'longEdge ) ) 0.20
    (("VA" 'longEdge ) ("VB" 'shortEdge)) 0.21
    (("VA" 'longEdge ) ("VB" 'longEdge ) ) 0.23
    (("VB" 'shortEdge) ("VA" 'shortEdge)) 0.20
    (("VB" 'shortEdge) ("VA" 'longEdge ) ) 0.21
    (("VB" 'shortEdge) ("VB" 'shortEdge)) 0.30
    (("VB" 'shortEdge) ("VB" 'longEdge ) ) 0.30
    (("VB" 'longEdge ) ("VA" 'shortEdge)) 0.21
    (("VB" 'longEdge ) ("VA" 'longEdge ) ) 0.23
    (("VB" 'longEdge ) ("VB" 'shortEdge)) 0.30
    (("VB" 'longEdge ) ("VB" 'longEdge ) ) 0.30
  )
)
( minCutClassSpacing "Via1"
  (( "cutClass" (("VA" 'shortEdge) ("VA" 'longEdge)
                  ("VB" 'shortEdge) ("VB" 'longEdge)) nil
      "cutClass" nil nil )
  'paraOverlap -0.2 'bothNegativePrls
)
(
  (("VA" 'shortEdge) ("VA" 'shortEdge)) 0.20
  (("VA" 'shortEdge) ("VA" 'longEdge ) ) 0.20
  (("VA" 'shortEdge) ("VB" 'shortEdge)) 0.20
  (("VA" 'shortEdge) ("VB" 'longEdge ) ) 0.22
  (("VA" 'longEdge ) ("VA" 'shortEdge)) 0.20
  (("VA" 'longEdge ) ("VA" 'longEdge ) ) 0.20
  (("VA" 'longEdge ) ("VB" 'shortEdge)) 0.21
  (("VA" 'longEdge ) ("VB" 'longEdge ) ) 0.24
  (("VB" 'shortEdge) ("VA" 'shortEdge)) 0.20
  (("VB" 'shortEdge) ("VA" 'longEdge ) ) 0.21
  (("VB" 'shortEdge) ("VB" 'shortEdge)) 0.30
  (("VB" 'shortEdge) ("VB" 'longEdge ) ) 0.30
  (("VB" 'longEdge ) ("VA" 'shortEdge)) 0.21
  (("VB" 'longEdge ) ("VA" 'longEdge ) ) 0.23
  (("VB" 'longEdge ) ("VB" 'shortEdge)) 0.30
  (("VB" 'longEdge ) ("VB" 'longEdge ) ) 0.30
)
) ;spacingTables
```

■ Via1

Virtuoso Technology Data Constraints Reference

Via Construction Constraints



PASS. Because 'bothNegativePrIs 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 (("VA" 'longEdge) ("VB" 'shortEdge))

Vertical PRL = -0.1 => spacing = 0.22 (("VA" 'shortEdge) ("VB" 'longEdge))

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).

minCutClassSpacing (Two layers)

```
spacingTables(  
  ( minCutClassSpacing tx_cutLayer1 tx_cutLayer2  
    (( "cutClass" (g_index ...) nil "cutClass" (g_index ...) nil)  
      [['sameNet | 'sameMetal] | ['exceptSameNet]]  
      ['spacingPr1 (f_pr1 f_spacing)]  
      ['centerToCenter]  
      ['paraOverlap f_overlap]  
      ['nonCutClassEdgeSpacing (f_length1 f_space1 f_length2 f_space2 ...)]  
      ['cutClassProfile  
        (( "cutClass" (g_index ...) nil "cutClass" (g_index ...) nil )  
          (g_cutClassProfileTable)  
        )  
      ]  
      ['nonZeroEnclosure tx_layer  
        ['minEnclosure f_minEnclosure]  
      ]  
      ['cutClassSizeBy ({f_size1 f_size2} ...)]  
      [f_default]  
    )  
    (g_table) ['manhattan]  
  )  
) ;spacingTables
```

Specifies the spacing between cut shapes of the same or of different classes on two different layers. The spacing requirement depends on the size of the cut shape and on the edge—long or short—from which the spacing is measured.

Optional parameters determine whether spacing is measured center-to-center or edge-to-edge, whether the constraint applies to a certain connectivity type, and whether the constraint applies only if the cut shapes have a parallel run length greater than or equal to the specified value.

Note: 'cutClassProfile is mutually exclusive with 'cutClassSizeBy.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<code>tx_cutLayer1</code>	The first cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>tx_cutLayer2</code>	The second cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>"cutClass" (g_index ...) nil "cutClass" (g_index ...) nil</code>	This identifies the index for <i>table</i>. The format of <i>g_index</i> is as follows: <pre>f_width (f_width f_length) t_className ((f_width f_length) 'shortEdge 'longEdge) (t_className 'shortEdge 'longEdge)</pre>

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

g_table

The spacing between the long edges (sides) and/or short edges (ends) of cuts, of the same class or of different classes.

The *table* row has the following format:

```
((f_edgeLength | t_className 'shortEdge |  
'longEdge) (f_edgeLength | t_className  
'shortEdge | 'longEdge)) f_spacing |  
(f_spacing ['centerToCenter | 'centerAndEdge |  
'centerAndEdgeNoPRL])
```

where,

- *f_edgeLength* is the width or length of the cut class.
- *t_className* is the name of the cut class and must correspond to a cut class defined in a [cutClasses](#) constraint.
- 'shortEdge and 'longEdge indicate the edge of the cut shape from which spacing is measured.
- *f_spacing* is the minimum spacing required between cut shapes.
- 'centerToCenter indicates that spacing is measured from the center of one cut shape to the center of another cut shape.
- 'centerAndEdge specifies that the maximum of all applicable spacing values must be measured center-to-center and the minimum of all applicable spacing values must be measured edge-to-edge.
- 'centerAndEdgeNoPRL specifies that the spacing between two cut shapes must be the greater of the following two values:
 - The maximum of all applicable values, which is measured center-to-center.
 - The minimum of all applicable values, which is measured edge-to-edge.

Type: A 2-D table specifying floating-point width or length and spacing values.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

'sameNet | 'sameMetal

The connectivity type. If connectivity type is not specified, the constraint applies to any connectivity type, including any two via cuts on the same cut layer with no common metal or net.

- 'sameNet: The constraint applies only if the via cuts are on the same net.
- 'sameMetal: The constraint applies only if the via cuts are on a contiguous same-metal shape.

Type: Boolean

'spacingPr1 (*f_pr1* *f_spacing*)

Specifies that two cuts, regardless of the cut class, with parallel run less than or equal to *f_pr1* in both directions should have a spacing greater than or equal to *f_spacing*.

'exceptSameNet

The constraint does not apply to via cuts on the same net.

Type: Boolean

'centerToCenter

The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'paraOverlap *f_overlap*

The constraint applies only if the parallel run length (prl) between the cut shapes is greater than or equal to this value. Positive and negative values are allowed.

If multiple `minCutClassSpacing` constraints are defined in an AND constraint group, the constraint is applied by evaluating the *overlap* values, specified in the ascending order, to locate the first *overlap* value for which the actual parallel run length is greater than or equal to this value. As a result, only one constraint in an AND constraint group applies. For example, in an AND constraint group with parallel run length values specified as -1.0, -0.05, 0, 0.05, 1, the constraint value that applies is determined as follows:

Actual <i>overlap</i> (prl)	Use constraint value for <i>overlap</i>
<-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

'nonCutClassEdgeSpacing (*f_length1* *f_space1*)

(ICADVM20.1 Only–95512) The spacing between a cut shape and a non-cut-class shape, such as a blockage, of length *length1* must be greater than or equal to *space1*.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
'cutClassProfile (( "cutClass" (g_index ...) nil "cutClass" (g_index ...) nil ) (g_cutClassProfileTable))
```

The cut shapes with the specified dimensions or class are expanded by a value equal to extension before spacing is measured.

The table has the following format:

```
((f_edgeLength | t_className
  'shortEdge | 'longEdge
)
(f_edgeLength | t_className
  'shortEdge | 'longEdge
)
) f_extension
```

This value is applied only if the parallel run length between the cut shapes is less than or equal to zero.

Type: A 2-D table specifying floating-point width or length values and the value by which the cut shapes need to be expanded.

```
'nonZeroEnclosure tx_layer
```

(ICADVM20.1 Only–95512) The constraint applies only if *cutLayer1* has *tx_layer* as the top layer and the shape on *cutLayer1* has a non-zero *layer* enclosure on all four sides.

This parameter applies only to the spacing checks between via cuts with parallel run length greater than zero.

Type: String (layer name) or Integer (layer number)

```
'minEnclosure f_minEnclosure
```

(ICADVM20.1 Only–95511) The constraint applies only if the via cut to which *'nonZeroEnclosure* applies has an enclosure greater than this value on all four sides.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'cutClassSizeBy ({*f_size1* *f_size2*} ...)

(ICADVM20.1 Only–95512) The constraint is applied after expanding all edges of the cut shape by this value. If there are n rows and m columns in constraint value table, this parameter should specify exactly $(n + m)$ extension values, such that the first n values specify extensions for the *cutLayer1* cut edges in the row indexes of the constraint value table, while the remaining m values specify extensions for the *cutLayer2* cut edges in the column indexes of the constraint value table.

f_default

The spacing value to be used when no table entry applies.

'manhattan

The constraint uses Manhattan distance, which allows a larger spacing at the corners.

By default, the constraint uses Euclidian measurement.

Type: Boolean

Examples

- [Example 1: minCutClassSpacing with paraOverlap](#)
- [Example 2: minCutClassSpacing with cutClass, cutClassProfile, sameNet, and paraOverlap](#)
- [Example 3: minCutClassSpacing with cutClass and cutClassSizeBy](#)
- [Example 4: minCutClassSpacing with cutClass and nonZeroEnclosure](#)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 1: minCutClassSpacing with paraOverlap

```
spacingTables (
  ( minCutClassSpacing "Via1" "Via2"
    (( "cutClass" ("VA" ("VB" 'shortEdge) ("VB" 'longEdge) "VC") nil
      "cutClass" ("VD" ("VE" 'shortEdge) ("VE" 'longEdge) "VF") nil )
    'paraOverlap 0.001
    0.15
  )
  (
    ((("VA" 'shortEdge) ("VD" 'shortEdge)) (0.20 'centerToCenter)
      ("VA" 'shortEdge) ("VD" 'longEdge ) ) (0.20 'centerToCenter)
      ("VA" 'shortEdge) ("VE" 'shortEdge)) 0.30
      ("VA" 'shortEdge) ("VE" 'longEdge ) ) 0.15
      ("VA" 'shortEdge) ("VF" 'shortEdge)) 0.15
      ("VA" 'shortEdge) ("VF" 'longEdge ) ) 0.15
      ("VA" 'longEdge ) ("VD" 'shortEdge)) (0.20 'centerToCenter)
      ("VA" 'longEdge ) ("VD" 'longEdge ) ) (0.20 'centerToCenter)
      ("VA" 'longEdge ) ("VE" 'shortEdge)) 0.30
      ("VA" 'longEdge ) ("VE" 'longEdge ) ) 0.15
      ("VA" 'longEdge ) ("VF" 'shortEdge)) 0.15
      ("VA" 'longEdge ) ("VF" 'longEdge ) ) 0.15
      ("VB" 'shortEdge) ("VD" 'shortEdge)) 0.30
      ("VB" 'shortEdge) ("VD" 'longEdge ) ) 0.30
      ("VB" 'shortEdge) ("VE" 'shortEdge)) 0.40
      ("VB" 'shortEdge) ("VE" 'longEdge ) ) 0.40
      ("VB" 'shortEdge) ("VF" 'shortEdge)) 0.30
      ("VB" 'shortEdge) ("VF" 'longEdge ) ) 0.30
      ("VB" 'longEdge ) ("VD" 'shortEdge)) 0.15
      ("VB" 'longEdge ) ("VD" 'longEdge ) ) 0.15
      ("VB" 'longEdge ) ("VE" 'shortEdge)) 0.40
      ("VB" 'longEdge ) ("VE" 'longEdge ) ) 0.15
      ("VB" 'longEdge ) ("VF" 'shortEdge)) 0.15
      ("VB" 'longEdge ) ("VF" 'longEdge ) ) 0.15
      ("VC" 'shortEdge) ("VD" 'shortEdge)) 0.15
      ("VC" 'shortEdge) ("VD" 'longEdge ) ) 0.15
      ("VC" 'shortEdge) ("VE" 'shortEdge)) 0.30
      ("VC" 'shortEdge) ("VE" 'longEdge ) ) 0.15
      ("VC" 'shortEdge) ("VF" 'shortEdge)) (0.50 'centerToCenter)
      ("VC" 'shortEdge) ("VF" 'longEdge ) ) (0.50 'centerToCenter)
      ("VC" 'longEdge ) ("VD" 'shortEdge)) 0.15
      ("VC" 'longEdge ) ("VD" 'longEdge ) ) 0.15
      ("VC" 'longEdge ) ("VE" 'shortEdge)) 0.30
      ("VC" 'longEdge ) ("VE" 'longEdge ) ) 0.15
      ("VC" 'longEdge ) ("VF" 'shortEdge)) (0.50 'centerToCenter)
      ("VC" 'longEdge ) ("VF" 'longEdge ) ) (0.50 'centerToCenter)
    )
  )
);spacingTables
```

To find the spacing between a VA via cut (0.3x0.3) and the short edge of a VE via cut (0.4x0.6):

- First, traverse the list of ordered row indexes to locate the entries that correspond to cut class VA.
- Then, traverse the list of ordered column indexes to locate the entry that matches "VE" 'shortEdge.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 2: minCutClassSpacing with cutClass, cutClassProfile, sameNet, and paraOverlap

```
spacingTables(  
  ( minCutClassSpacing "Via1" "Via2"  
    (( "cutClass" ("VD" ("VE" 'longEdge) ("VE" 'shortEdge) "VF") nil  
      "cutClass" ("V2D" ("V2E" 'shortEdge) ("V2E" 'longEdge) "V2F") nil )  
    'sameNet  
    'paraOverlap 0.001  
    'cutClassProfile  
      (( "cutClass" (("VE" 'shortEdge) ("VE" 'longEdge)) nil  
        "cutClass" ("V2D" ("V2E" 'shortEdge) ("V2E" 'longEdge) "V2F") nil  
      )  
      (  
        (( "VE" 'shortEdge) ("V2D" 'shortEdge)) 0.02  
        (( "VE" 'shortEdge) ("V2D" 'longEdge )) 0.02  
        (( "VE" 'shortEdge) ("V2E" 'shortEdge)) 0.00  
        (( "VE" 'shortEdge) ("V2E" 'longEdge )) 0.00  
        (( "VE" 'shortEdge) ("V2F" 'shortEdge)) 0.02  
        (( "VE" 'shortEdge) ("V2F" 'longEdge )) 0.02  
        (( "VE" 'longEdge ) ("V2D" 'shortEdge)) 0.00  
        (( "VE" 'longEdge ) ("V2D" 'longEdge )) 0.00  
        (( "VE" 'longEdge ) ("V2E" 'shortEdge)) 0.00  
        (( "VE" 'longEdge ) ("V2E" 'longEdge )) 0.02  
        (( "VE" 'longEdge ) ("V2F" 'shortEdge)) 0.00  
        (( "VE" 'longEdge ) ("V2F" 'longEdge )) 0.00  
      )  
    )  
  )  
  0.15  
)  
(  
  (( "VD" 'shortEdge) ("V2D" 'shortEdge)) 0.2  
  (( "VD" 'shortEdge) ("V2D" 'longEdge )) 0.2  
  (( "VD" 'shortEdge) ("V2E" 'shortEdge)) 0.3  
  (( "VD" 'longEdge ) ("V2D" 'shortEdge)) 0.2  
  (( "VD" 'longEdge ) ("V2D" 'longEdge )) 0.2  
  (( "VD" 'longEdge ) ("V2E" 'shortEdge)) 0.3  
  (( "VE" 'longEdge ) ("V2E" 'shortEdge)) 0.4  
  (( "VE" 'shortEdge) ("V2D" 'shortEdge)) 0.3  
  (( "VE" 'shortEdge) ("V2D" 'longEdge )) 0.3  
  (( "VE" 'shortEdge) ("V2E" 'shortEdge)) 0.4  
  (( "VE" 'shortEdge) ("V2E" 'longEdge )) 0.4  
  (( "VE" 'shortEdge) ("V2F" 'shortEdge)) 0.3  
  (( "VE" 'shortEdge) ("V2F" 'longEdge )) 0.3  
  (( "VF" 'shortEdge) ("V2E" 'shortEdge)) 0.3  
  (( "VF" 'shortEdge) ("V2F" 'shortEdge)) 0.5  
  (( "VF" 'shortEdge) ("V2F" 'longEdge )) 0.5  
  (( "VF" 'longEdge ) ("V2E" 'shortEdge)) 0.3  
  (( "VF" 'longEdge ) ("V2F" 'shortEdge)) 0.5  
  (( "VF" 'longEdge ) ("V2F" 'longEdge )) 0.5  
)  
) ;spacingTables
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The 'cutClassProfile parameter uses its own 2-D table, as shown below:

			V2D		V2E		V2F	
			SE	LE	SE	LE	SE	LE
			50	50	50	100	100	100
V2E	SE	50	0.02	0.02	0.0	<i>0.0</i>	0.02	0.02
	LE	100	0.0	0.0	<i>0.0</i>	0.02	0.0	0.0

For square cuts, V2D and V2F, the LE ('longEdge) and SE ('shortEdge) entries must be equal, and for rectangular cuts, only LE/LE and SE/SE entries are valid. Therefore, the two entries in italics are ignored. Additionally, only the dimensions of rectangular cut shapes are used to index rows.

Example 3: minCutClassSpacing with cutClass and cutClassSizeBy

The spacing between cut shapes on layers Via1 and Via2 is defined as follows:

	(Via2)	W3	L3	W4	L4
(Via1)		0.1	0.1	0.2	0.2
W1	0.1	0.15	0.15	0.20	0.20
L1	0.1	0.15	0.15	0.20	0.20
W2	0.1	0.15	0.15	0.20	0.20
L2	0.2	0.20	0.20	0.25	0.25

The 'cutClassSizeBy value applied to various cut edges is as follows:

	Via1				Via2			
W/L	0.1	0.1	0.1	0.2	0.1	0.1	0.2	0.2
Size by	0.01	0.01	0.01	0.01	0.0	0.0	0.01	0.01

- When measuring spacing from VA on Via1:
 - ❑ To VA on Via2, the edges of the via cut on Via1 should be extended by 0.01.
 - ❑ To VB on Via2, the edges of the via cuts on both layers should be extended by 0.01.
- When measuring spacing from VB on Via1:

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

- ❑ To VA on Via2, the edges of the via cut on Via1 should be extended by 0.1.
- ❑ To VB on Via2, the edges of the via cuts on both layers should be extended by 0.01.

```
spacingTables(
  ( minCutClassSpacing "Via1" "Via2"
    (( "cutClass" ("VA" ("VB" 'shortEdge) ("VB" 'longEdge)) nil
      "cutClass" ("VA" "VB") nil )
      'cutClassSizeBy (0.01 0.01 0.01 0.01 0.0 0.0 0.01 0.01)
    )
    (
      (("VA" 'shortEdge) ("VA" 'shortEdge)) 0.15
      (("VA" 'shortEdge) ("VA" 'longEdge )) 0.15
      (("VA" 'shortEdge) ("VB" 'shortEdge)) 0.15
      (("VA" 'shortEdge) ("VB" 'longEdge )) 0.20
      (("VA" 'longEdge ) ("VA" 'shortEdge)) 0.15
      (("VA" 'longEdge ) ("VA" 'longEdge )) 0.15
      (("VA" 'longEdge ) ("VB" 'shortEdge)) 0.15
      (("VA" 'longEdge ) ("VB" 'longEdge )) 0.20
      (("VB" 'shortEdge) ("VA" 'shortEdge)) 0.20
      (("VB" 'shortEdge) ("VA" 'longEdge )) 0.20
      (("VB" 'shortEdge) ("VB" 'shortEdge)) 0.20
      (("VB" 'shortEdge) ("VB" 'longEdge )) 0.25
      (("VB" 'longEdge ) ("VA" 'shortEdge)) 0.20
      (("VB" 'longEdge ) ("VA" 'longEdge )) 0.20
      (("VB" 'longEdge ) ("VB" 'shortEdge)) 0.20
      (("VB" 'longEdge ) ("VB" 'longEdge )) 0.25
    )
  )
) ;spacingTables
```

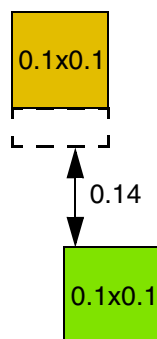
■ Via2

■ Via1

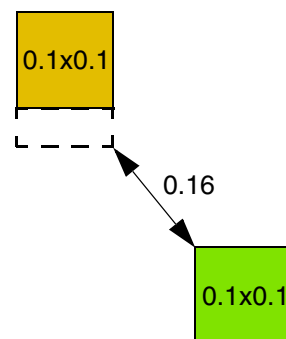
❑ Sized edge

Virtuoso Technology Data Constraints Reference

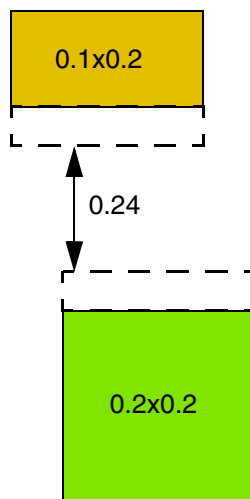
Via Construction Constraints



a) FAIL. The spacing between the via cuts is only 0.14 (<0.15). The VA cut shape on Via1 is sized by 0.01 before spacing is measured.



b) PASS. The spacing between the via cuts is 0.16 (>0.15). The VA cut shape on Via1 is sized by 0.01 before spacing is measured.



c) FAIL. The spacing between the via cuts is only 0.24 (<0.25). The VB cut shapes on both Via1 and Via2 are sized by 0.01 before spacing is measured.

Virtuoso Technology Data Constraints Reference

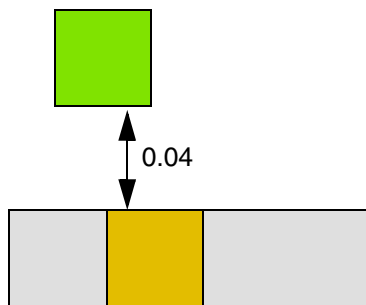
Via Construction Constraints

Example 4: minCutClassSpacing with cutClass and nonZeroEnclosure

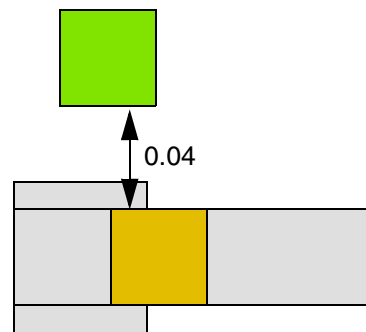
The constraint applies only if the Via1 via cut has non-zero Metal2 enclosure.

```
spacingTables (
  ( minCutClassSpacing "Via1" "Via2"
    (( "cutClass" ("VA" ("VB" 'shortEdge) ("VB" 'longEdge)) nil
      "cutClass" ("VA" ("VB" 'shortEdge) ("VB" 'longEdge)) nil )
    'exceptSameNet
    'paraOverlap 0.001
    'nonZeroEnclosure "Metal2"
  )
  (
    (("VA" 'shortEdge) ("VA" 'shortEdge)) 0.15
    (("VA" 'shortEdge) ("VA" 'longEdge )) 0.15
    (("VA" 'shortEdge) ("VB" 'shortEdge)) 0.15
    (("VA" 'shortEdge) ("VB" 'longEdge )) 0.20
    (("VA" 'longEdge ) ("VA" 'shortEdge)) 0.15
    (("VA" 'longEdge ) ("VA" 'longEdge )) 0.15
    (("VA" 'longEdge ) ("VB" 'shortEdge)) 0.15
    (("VA" 'longEdge ) ("VB" 'longEdge )) 0.20
    (("VB" 'shortEdge) ("VA" 'shortEdge)) 0.20
    (("VB" 'shortEdge) ("VA" 'longEdge )) 0.20
    (("VB" 'shortEdge) ("VB" 'shortEdge)) 0.20
    (("VB" 'shortEdge) ("VB" 'longEdge )) 0.25
    (("VB" 'longEdge ) ("VA" 'shortEdge)) 0.20
    (("VB" 'longEdge ) ("VA" 'longEdge )) 0.20
    (("VB" 'longEdge ) ("VB" 'shortEdge)) 0.20
    (("VB" 'longEdge ) ("VB" 'longEdge )) 0.25
  )
) ;spacingTables
```

■ Via2
■ Via1
□ Metal2



a) The constraint does not apply because the top edge of the Via1 via cut does not have a Metal2 enclosure.



b) FAIL. The top edge of the Via1 via cut has a non-zero Metal2 enclosure and this edge has a neighboring via cut with which it has a parallel run length greater than zero. However, the spacing between the two via cuts is only 0.04 (>0.15).

minDirectionalSpacing

```

spacings (
    ( minDirectionalSpacing tx_layer
      {'horizontal' | 'vertical'}
      'cutPr1 g_cutPr1
      'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
      'otherCutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
      ['pr1 f_pr1 'within f_within
        ['excludeCutEdge]
        ['minNumCuts g_minNumCuts]
        [groupName t_groupName]
      ]
      g_spacing
    )
) ;spacings

```

(ICADVM20.1 Only – 95511) Specifies the directional spacing between two cuts, one on `cutClass` and the other on `otherCutClass`, when the cuts have more than `cutPr1` parallel run length.

This constraint applies selectively if a cut edge along the orthogonal direction of the spacing of a cut in the specified `cutClass` has a neighboring cut in the specified `otherCutClass` in the specified `within` distance when the parallel run length is more than the `pr1` value.

Values

<code>tx_layer</code>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>g_spacing</code>	The spacing between the cut shapes must be greater than or equal to this value.

Parameters

<code>'horizontal' 'vertical'</code>	The direction in which the cut spacing applies. Type: Boolean
<code>'cutPr1 g_cutPr1'</code>	The constraint applies when the parallel run length of the two cuts is greater than the given value.
<code>'cutClass {f_cutWidth (f_cutWidth f_cutLength) t_name}'</code>	

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The constraint applies to the cut class specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

```
'otherCutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
```

The constraint applies to this additional cut class specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

```
'prl f_prl
```

The constraint applies only if a cut edge in the orthogonal direction of a cut of `cutClass` has a neighboring cut of `otherCutClass` with the parallel run length greater than `prl` on the same side.

```
'within f_within
```

The constraint applies only if a cut edge in the orthogonal direction of a cut of `cutClass` has a neighboring cut of `otherCutClass` within this value on the same side.

```
'excludeCutEdge
```

Separate neighbor cut search windows are formed from cut corners based on the given values in *f_prl* and *f_within*. Each window is checked.

```
'minNumCuts g_minNumCuts
```

The constraint applies only if the neighbor search window has at least the number of cuts specified.

```
'groupName t_groupName
```

The constraint applies if the neighbor cut of `otherCutClass` belongs to the specified via group.

Examples

- [Example 1: minDirectionalSpacing with prl, within, and groupName](#)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

■ Example 2: minDirectionalSpacing with excludeCutEdge and minNumCuts

Example 1: minDirectionalSpacing with prl, within, and groupName

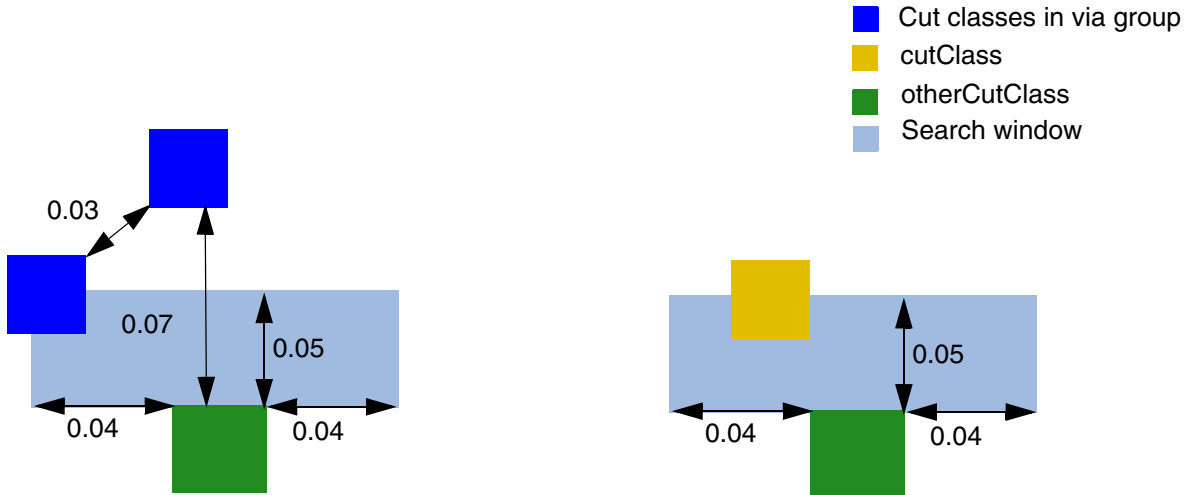
Consider the following sample constraints.

```
spacings(  
  (minDirectionalSpacing "Via1"  
    'vertical 'cutPrl -0.02  
    'cutClass "VB" 'otherCutClass "VA"  
    'prl -0.04 'within 0.05  
    0.07)  
  (minDirectionalSpacing "Via1"  
    'vertical 'cutPrl -0.02  
    'cutClass "VB" 'otherCutClass "VA"  
    'groupName "VGA" 'prl -0.04 'within 0.05  
    0.08)
```

Virtuoso Technology Data Constraints Reference

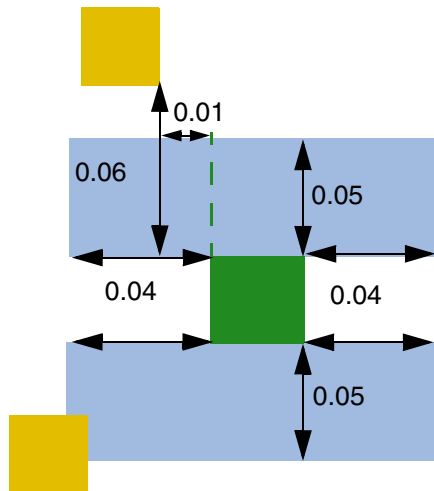
Via Construction Constraints

) ;spacings

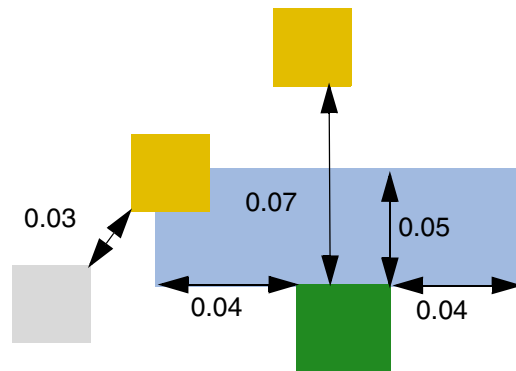


a. FAIL. The constraint violates the 0.08 spacing requirement because the neighboring cut belongs to the via group VGA.

b. FAIL. A neighbor cut of the VA cutClass overlaps with the VB search window whereas a cut spacing of 0.07 is required.



c. PASS. The constraint does not apply. It applies only when the cut and the neighbor cut are on the same side.



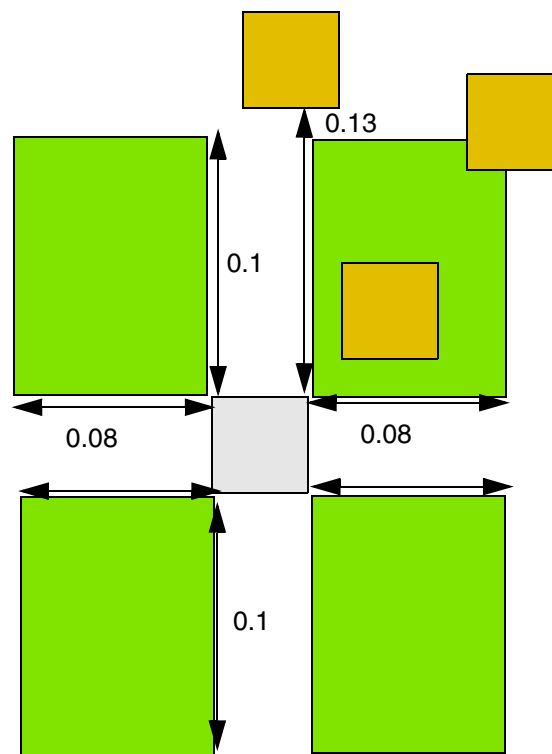
d. PASS. The first constraint applies. All conditions are met. The second constraint does not apply.

Example 2: *minDirectionalSpacing* with *excludeCutEdge* and *minNumCuts*

Consider the following sample constraint.

```
spacings (
  (minDirectionalSpacing "Vial"
    'vertical
    'cutPRL 0
    'cutClass "VA"
    'otherCutClass "VB"
    'excludeCutEdge
    'minNumCuts 2
    'prl -0.08
    'within 0.1
    0.08
  )
) ;spacings
```

Search window
 VA
 VB



Violation. The four search windows are formed from the cut corners and each of the windows would be checked for neighbor cuts of VA individually. It is a violation because the upper right window has two VA cuts and there is another VA cut on top of VB with a spacing of 0.13 (< 0.15).

minEndOfLineCutSpacing

```
orderedSpacings (
    ( minEndOfLineCutSpacing tx_cutLayer tx_metalLayer
      ['cutClass { f_width | (f_width f_length) | t_name }
      ['toCutClass
        (
          ({ f_width1 | (f_width1 f_lenght1) | t_name1 } (f_cS1 f_cS2))
          ...
        )
      ]
    ]
    'width f_width
    'prl f_prl
    'enclosure (f_eolEnclosure f_otherEnclosure)
    'extension (f_sideExt f_backwardExt)
    'spanLength f_spanLength
    (f_cutSpacing1 f_cutSpacing2)
  )
) ;orderedSpacings
```

Defines the minimum spacing required between via cuts that are near the end-of-line edges of the enclosing metal shape, based on the parallel run length (*prl*) between the via cuts.

The placement of the via cut within the enclosing metal shape, together with the end-of-line width and span length of the enclosing metal shape, determines the via cut edges that must satisfy the spacing requirement. For example, in the figure below:

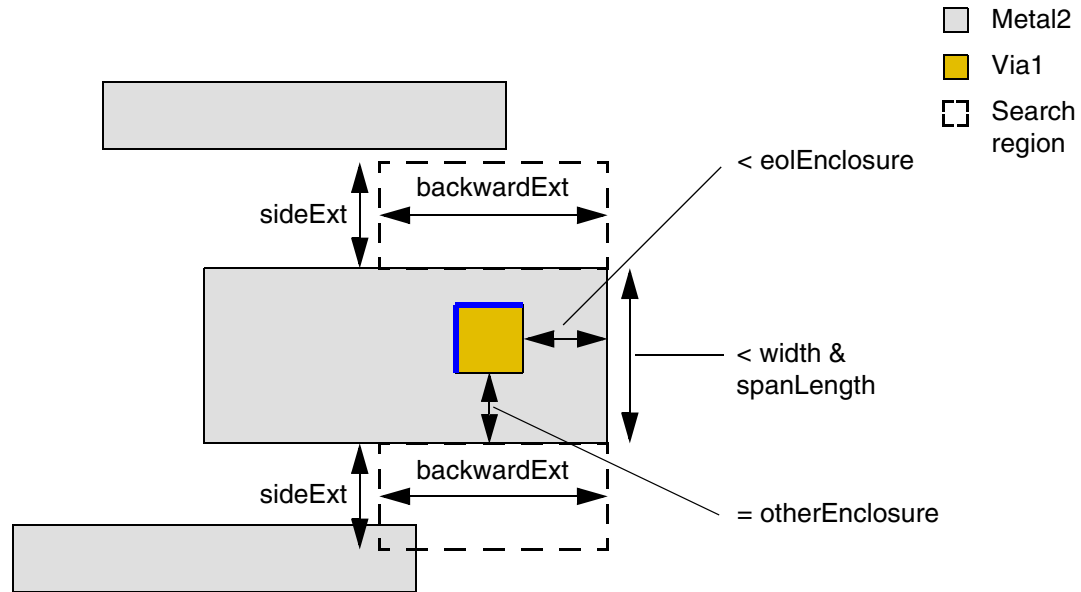
- The end-of-line width is less than *eolWidth* and the span length is less than *spanLength*.
- The end-of-line edge enclosure is less than *eolEnclosure* and the enclosure of the bottom cut edge is exactly equal to *otherEnclosure*.

Because the span length is less than *spanLength*, a neighboring wire must overlap with only the search region along the edge that satisfies *otherEnclosure* (and not with the other search region that is along the opposite edge)—a condition that is satisfied.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The cut spacing applies to the vertical blue edge of the via cut because it is opposite the end-of-line edge and to the horizontal blue edge of the via cut because it is opposite the edge that satisfies *otherEnclosure*.



In the next example shown in the figure below:

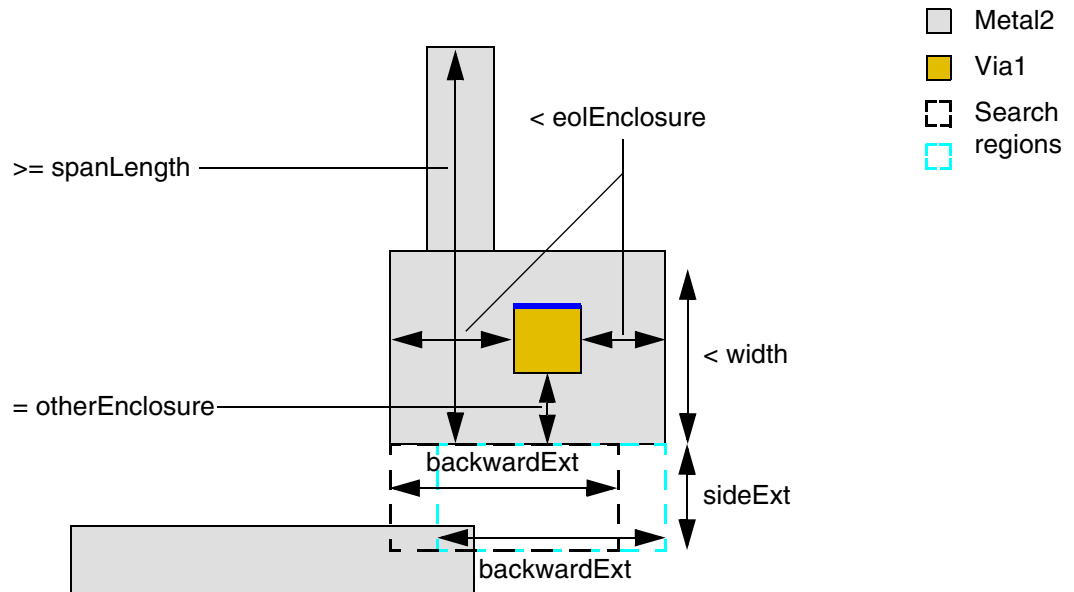
- The end-of-line width is less than *eolWidth* and the span length is greater than or equal to *spanLength*.
- The end-of-line edge enclosure is less than *eolEnclosure* and the enclosure of the bottom cut edge is exactly equal to *otherEnclosure*.

Because the via cut is within *eolEnclosure* of both end-of-line edges, the two search regions overlap, with one extending backward from the left end-of-line edge and the other extending backward from the right end-of-line edge, as shown below. This also means that there is no cut edge opposite the end-of-line edge to which the cut spacing applies. Therefore, the only edge to which the cut spacing applies is the horizontal blue edge opposite the edge that satisfies *otherEnclosure*. Note that a neighboring wire overlaps with both the search

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

regions—a condition that must be satisfied when span length is greater than or equal to *spanLength*.

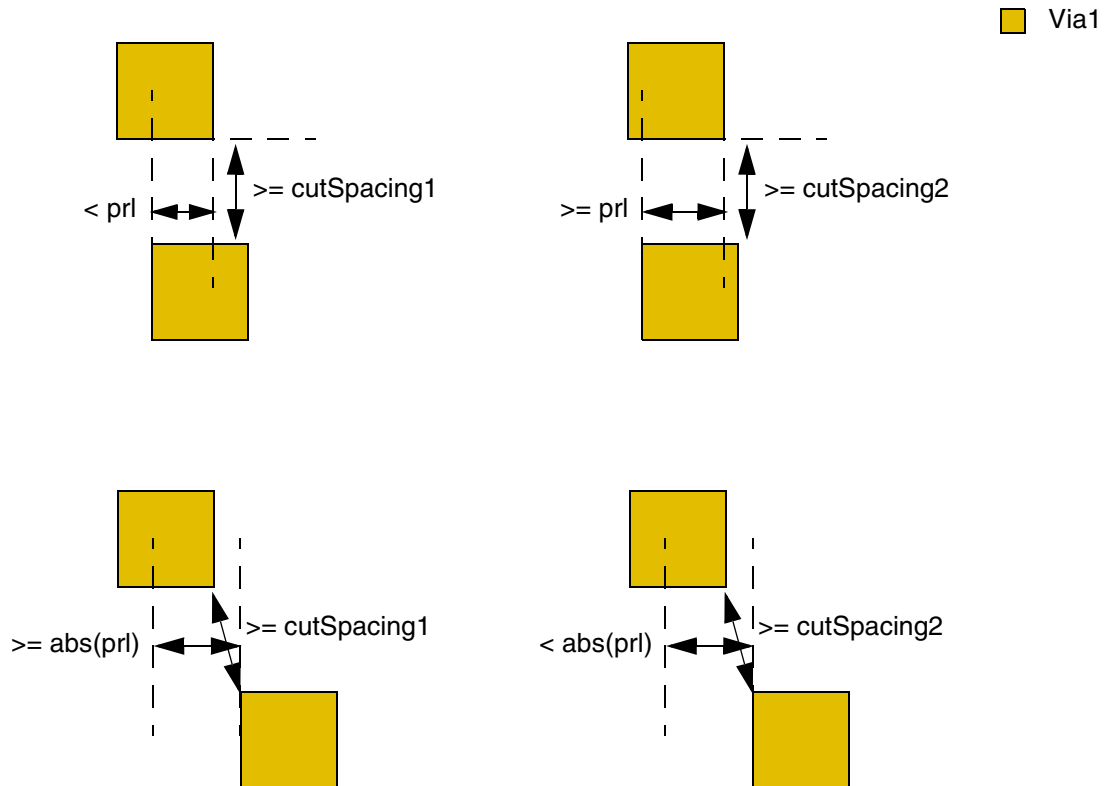


If a cut layer has cut classes, only one `minEndOfLineCutSpacing` constraint should be defined for each cut class on that cut layer. If the cut layer does not have cut classes, only one `minEndOfLineCutSpacing` constraint should be defined for that layer.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Cut spacing based on the parallel run length between the via cuts is measured as shown below:



Values

<code>tx_cutLayer</code>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>tx_metalLayer</code>	The metal layer above the cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>f_cutSpacing1</code> <code>f_cutSpacing2</code>	The minimum cut spacing required based on the parallel run length (<code>prl</code>) between the two via cuts (<code>cutSpacing1</code> is typically less than <code>cutSpacing2</code>).

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

`'cutClass { f_width | (f_width f_length) | t_name }`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- `f_width`: Width
- `f_length`: Length
- `t_name`: Name of the cut class

`'toCutClass (({ f_width1 | (f_width1 f_length1) | t_name1 }
(f_cS1 f_cS2)) ...)`

A list of cut classes and the required spacing from the named cut class to each cut class in the list.

`'width f_width`

The constraint applies only if the end-of-line width of the enclosing metal shape is less than this value.

`'prl f_prl`

The spacing between the two via cuts must be greater than or equal to:

- `cutSpacing1` if the parallel run length between the two via cuts is less than `prl`.
- `cutSpacing2` if the parallel run length between the two via cuts is greater than or equal to `prl`.

Both positive and negative values are allowed for `prl`.

`'enclosure (f_eolEnclosure f_otherEnclosure)`

The constraint applies only if the enclosure on the end-of-line edge is less than `eolEnclosure` and the enclosure on a cut edge orthogonal to the end-of-line edge is exactly equal to `otherEnclosure`.

`'extension (f_sideExt f_backwardExt)`

The two values together specify a search region that extends `sideExt` along the end-of-line edge with enclosure less than `eolEnclosure`, away from the wire, and `backwardExt` along an orthogonal edge.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'spanLength f_spanLength`

The span length of the enclosing metal shape.

- If the span length is less than this value, the constraint applies only if a neighboring wire overlaps with the search region along the edge that satisfies *otherEnclosure* (and not with the other search region that is along the opposite edge).
- If the span length is greater than or equal to this value, a neighboring wire must overlap with the search regions along the edge on which the via cut has enclosure exactly equal to *otherEnclosure*.

Example

The cut spacing required between Via1 via cuts enclosed by a Metal2 shape is as follows:

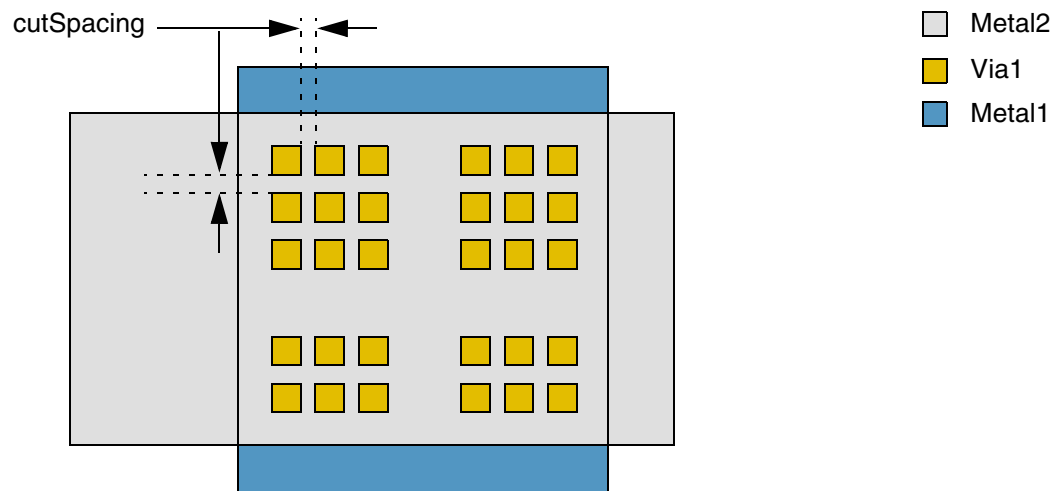
- (0.5 0.6) between two VA via cuts
- (0.2 0.2) between VA and VB via cuts
- (0.3 0.3) between VA and VC via cuts

```
ordered spacings (
  ( minEndOfLineCutSpacing "Via1" "Metal2"
    'cutClass "VA"
    'toCutClass (("VB" (0.2 0.2)) ("VC" (0.3 0.3)))
    'width 0.3
    'prl 0
    'enclosure (0.2 0.3)
    'extension (0.4 0.4)
    'spanLength 0.3
    (0.5 0.6)
  )
) ;ordered spacings
```

minLargeViaArrayCutSpacing

```
spacings (
    ( minLargeViaArrayCutSpacing tx_cutLayer
      ['minNumCuts x_minNumCuts]
      ['maxNumCuts x_maxNumCuts]
      ['applyPerDirection]
      f_spacing
    )
) ;spacings
```

Specifies the minimum spacing between via cuts in a large via cut array. Optionally, the constraint applies only within via cut arrays that satisfy the specified minimum and maximum requirements for via cuts.



Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between via cuts in a large via cut array must be greater than or equal to this value.

Parameters

```
'minNumCuts x_minNumCuts
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The constraint applies only if the number of rows and columns in the via cut array is greater than or equal to this value.

'maxNumCuts *x_maxNumCuts*

The constraint applies only if the number of rows and columns in the via cut array is less than or equal to this value.

'applyPerDirection The constraint applies separately in each direction between columns and rows that meet the values specified with minNumCuts and maxNumCuts.

Type: Boolean

Examples

- Example 1a: minLargeViaArrayCutSpacing with minNumCuts and maxNumCuts
- Example 1b: minLargeViaArrayCutSpacing with minNumCuts and maxNumCuts
- Example 2: minLargeViaArrayCutSpacing with applyPerDirection

Virtuoso Technology Data Constraints Reference

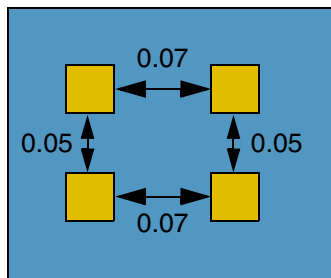
Via Construction Constraints

Example 1a: *minLargeViaArrayCutSpacing* with *minNumCuts* and *maxNumCuts*

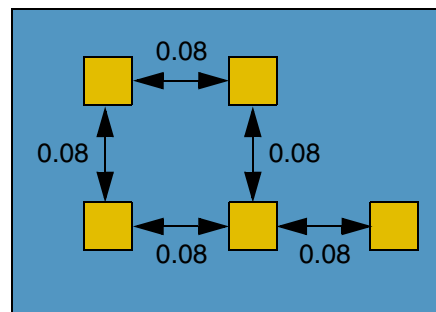
The spacing between the via cuts in a 2x2 via cut array must be greater than or equal to 0.07.

```
spacings(  
  ( minLargeViaArrayCutSpacing "Via1"  
    'minNumCuts 2 'maxNumCuts 2  
    0.07  
  )  
); spacings
```

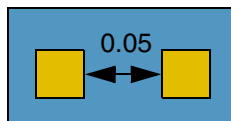
■ Via1
■ Metal1



a) FAIL. Though this is a 2x2 via cut array, the spacing between via cuts in the vertical direction is only 0.05 (<0.07).



b) PASS. The constraint applies to the 2x2 via cut array and the spacing between the via cuts in this array is 0.08 (>0.07).



c) The constraint does not apply because this is a 1x2 via cut array (and not 2x2).

Virtuoso Technology Data Constraints Reference

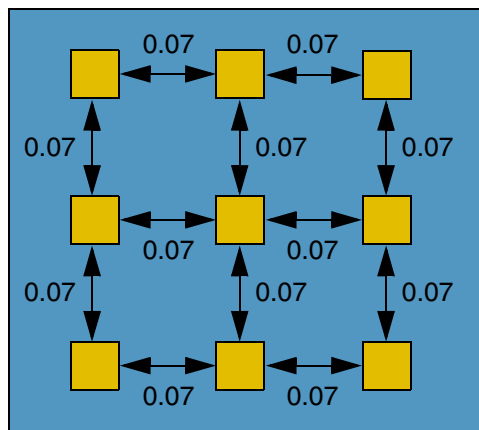
Via Construction Constraints

Example 1b: *minLargeViaArrayCutSpacing* with *minNumCuts* and *maxNumCuts*

The spacing between the via cuts in 2x2 and 3x3 via cut arrays must be greater than or equal to 0.07.

```
spacings(  
  ( minLargeViaArrayCutSpacing "Via1"  
    'minNumCuts 2 'maxNumCuts 3  
    0.07  
  )  
); spacings
```

■ Via1
■ Metal1



PASS. This is a 3x3 via cut array and the spacing between each pair of via cuts is 0.07.

Virtuoso Technology Data Constraints Reference

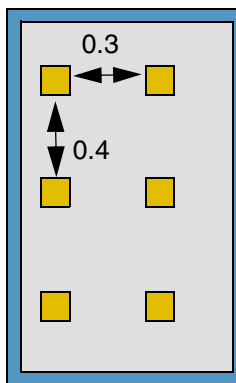
Via Construction Constraints

Example 2: *minLargeViaArrayCutSpacing* with *applyPerDirection*

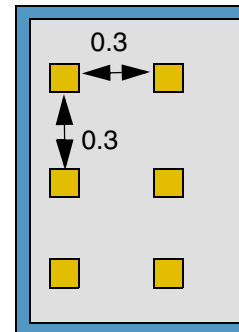
The spacing between the via cuts in 3x3 via cut arrays must be greater than or equal to 0.35, and the constraint applies separately in each direction.

```
spacings(  
  ( minLargeViaArrayCutSpacing "Via1"  
    'minNumCuts 3 'applyPerDirection  
    0.35  
  )  
) ;spacings
```

■ Via1
■ Metal1
■ Metal2



PASS. In the vertical direction, there are three cut rows. The constraint applies in this direction and $0.4 > 0.35$ is met. In the horizontal direction, the constraint does not apply because there are two cut columns.



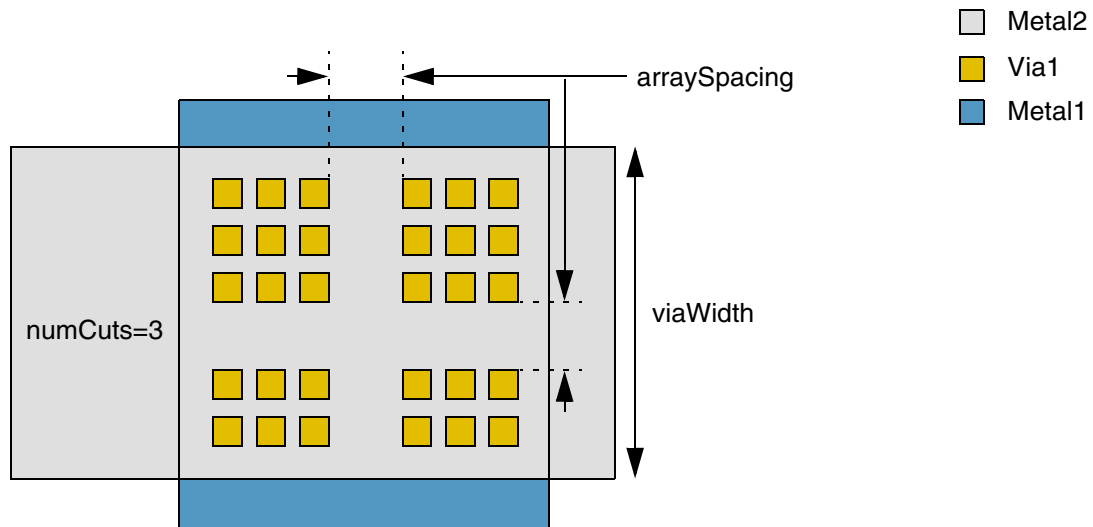
FAIL. In the vertical direction, the constraint applies because there are three cut rows but the constraint is not met because $0.3 < 0.4$. It would have passed if *applyPerDirection* was not specified.

minLargeViaArraySpacing

```
spacingTables (
  ( minLargeViaArraySpacing tx_cutLayer
    (( "numCuts" nil nil )
     ['cutClass { f_width | (f_width f_length) | t_name }]
     ['paraOverlap]
     ['intraArrayCutSpacing f_cutSpacing]
     [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum spacing (*arraySpacing*) between via cut arrays as a function of the minimum dimension of the array (*numCuts*).

A via cut array can be truncated if it does not fit within a wire, as shown below. In such an event, the large via array-to-array spacing is still used to place the truncated via cut array.



Only one `minLargeViaArraySpacing` constraint can be specified for a given layer, although multiple spacing values can be specified based on different array sizes. Additionally, you can use the `minLargeViaArrayWidth` constraint to specify if the constraint applies only to via cut arrays on wires with width greater than or equal to the *viaWidth*.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<code>tx_cutLayer</code>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>"numCuts" nil nil</code>	This identifies the index for <i>table</i> .
<code>g_table</code>	The format of a 1-D table is as follows: <pre>(x_numCuts f_spacing ...)</pre> where, <i>x_numCuts</i> represents the number of via cuts and <i>f_spacing</i> represents the minimum spacing that applies between via cut arrays when the number of via cuts in an array is greater than or equal to the corresponding index. Type: A 1-D table specifying the number of via cuts and floating-point spacing values.

Parameters

<code>'cutClass { f_width (f_width f_length) t_name }</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class
<code>'paraOverlap</code>	The constraint applies only if the via cut arrays have a non-zero parallel overlap. Type: Boolean
<code>'intraArrayCutSpacing f_cutSpacing</code>	The via cuts that are spaced by less than this value are considered to be part of the same via cut array.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

f_default The array spacing value to be used when no table entry applies.

Examples

- Example 1: minLargeViaArraySpacing
- Example 2: minLargeViaArraySpacing with intraArrayCutSpacing

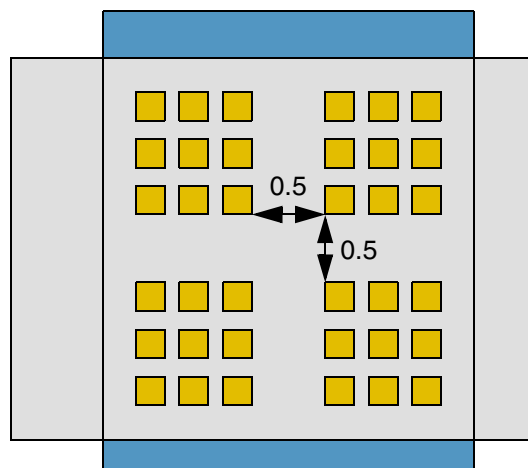
Example 1: minLargeViaArraySpacing

The minimum distance between via cut arrays in a large via must be as follows:

- 0.4 in a 2x2 via array
- 0.5 in a 3x3 via array
- 0.6 in a 4x4 via array
- 0.7 in a 5x5 via array

```
spacingTables(  
  ( minLargeViaArraySpacing "Via1"  
    (( "numCuts" nil nil ) 'intraArrayCutSpacing 0.1)  
    (  
      2    0.4  
      3    0.5  
      4    0.6  
      5    0.7  
    )  
  )  
) ;spacingTables
```

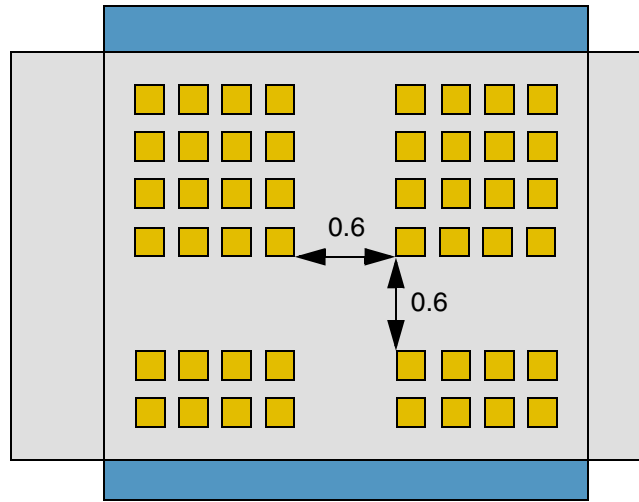
□ Metal2
■ Via1
■ Metal1



a) PASS. For a 3x3 via array, the required array spacing is at least 0.5, a condition that is satisfied.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints



b) PASS. For a 4x4 via array, the required array spacing is at least 0.6, a condition that is satisfied.

Example 2: minLargeViaArraySpacing with intraArrayCutSpacing

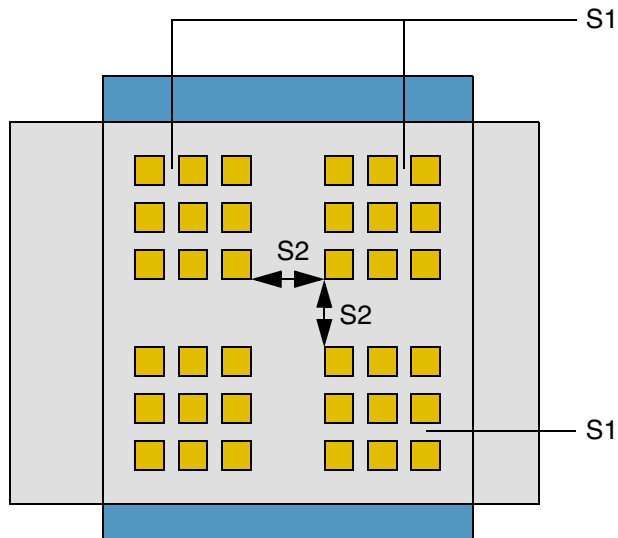
The minimum distance between via cut arrays in a large via must be as follows:

- v1 in a 1x1 via array
- v2 in a 2x2 via array
- v3 in a 3x3 via array
- v4 in a 4x4 via array

The spacing between the via cuts in each array must be less than SC for the via cuts to be considered as part of the same array.

```
spacingTables (
  ( minLargeViaArraySpacing "Vial"
    (( "numCuts" nil nil )
      'intraArrayCutSpacing SC
    )
    (
      1      v1
      2      v2
      3      v3
      4      v4
    )
  )
) ;spacingTables
```

Metal2
 Via1
 Metal1



PASS. The constraint is met if $S1 < SC$ and $S2 \geq v3$.

minLargeViaArrayWidth

```

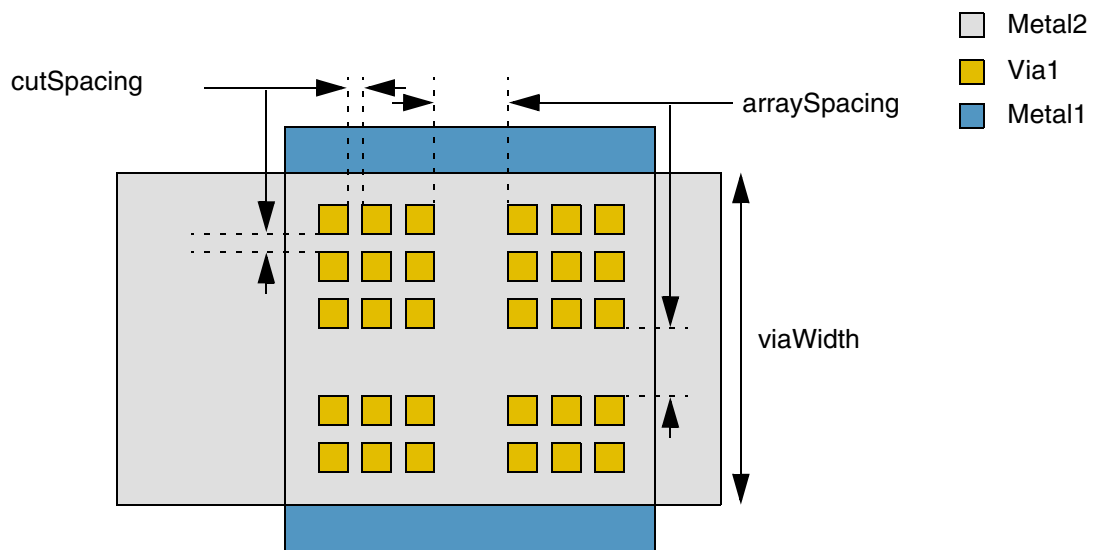
spacings (
  ( minLargeViaArrayWidth tx_cutLayer
    ['cutClass { f_width | (f_width f_length) | t_name }']
    ['paraOverlap']
    f_viaWidth
  )
) ;spacings

```

Specifies the minimum width of the intersection of two metal wires that are connected by the cut layer. This minimum width applies to both top and bottom metal layers in a via.

The `minLargeViaArrayWidth` and `minLargeViaArraySpacing` constraints together determine how via cut arrays are built. If `minLargeViaArrayWidth` is not specified, the array-to-array spacing (*arraySpacing*) specified by `minLargeViaArraySpacing` constraint is followed when a via cut array is built.

The via-cut-to-via-cut spacing (*cutSpacing*) is specified by the `minLargeViaArrayCutSpacing` constraint.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_viaWidth</i>	The width of the via must be greater than or equal to this value.

Parameters

<code>'cutClass { f_width (f_width f_length) t_name }</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class
<code>'paraOverlap</code>	The constraint applies only if the via cut arrays have a non-zero parallel overlap. Type: Boolean

minNeighborViaSpacing

```
spacings(  
    ( minNeighborViaSpacing tx_cutLayer  
      'within f_within  
      ['centerToCenter]  
      f_spacing  
    )  
); spacings
```

Specifies the minimum spacing between two cut shapes on the given cut layer if both cut shapes have a common neighboring cut shape within the given distance.

Values

<i>tx_cutLayer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the cut shapes must be greater than or equal to this value.

Parameters

'within <i>f_within</i>	The distance between a cut shape and the common neighboring cut shape must be less than this value.
'centerToCenter	The distance and spacing are measured center-to-center. Otherwise, the distance and spacing are measured edge-to-edge. Type: Boolean

Virtuoso Technology Data Constraints Reference

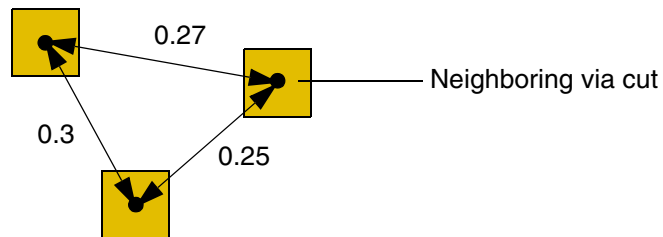
Via Construction Constraints

Example

The minimum spacing between two via cuts must be 0.2 if they have a common neighboring via cut at a distance less than 0.3.

```
spacings(  
  ( minNeighborViaSpacing "Via1"  
    'within 0.3  
    'centerToCenter  
    0.2  
  )  
) ;spacings
```

■ Via1

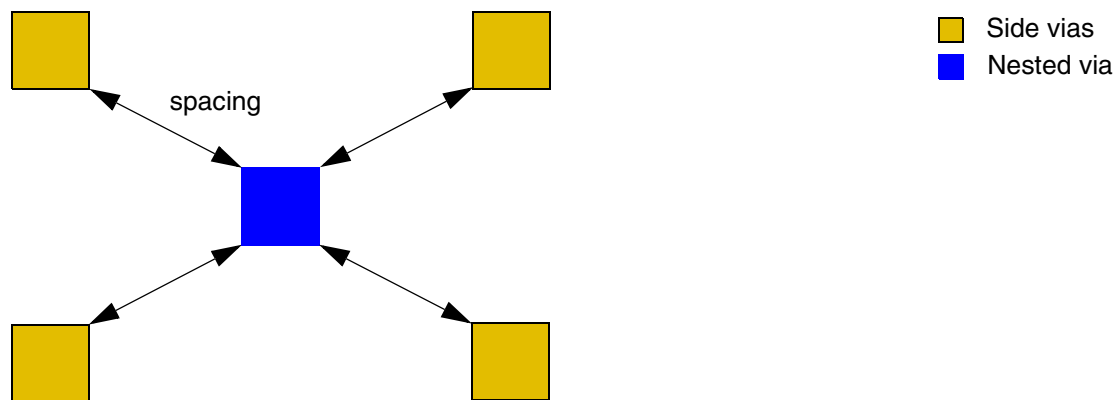


PASS. The two via cuts have a neighboring via cut at a distance less than 0.3 away and are spaced 0.3 apart (>0.2).

minNestedViaSpacing

```
spacings (  
    ( minNestedViaSpacing tx_cutLayer  
      'maxSideViaNeighbors n_maxSideViaNeighbors  
      ['maxSideViasWithNeighbors n_maxSideViasWithNeighbors]  
      f_spacing  
    )  
); spacings
```

(ICADVM20.1 Only – 95511) Specifies the minimum spacing between two nested vias and side vias on the given cut layer. A nested via has vias on all four corners, each with zero PRL. The vias in corners are referred to as side vias.



Values

<i>tx_cutLayer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between the cut shapes must be greater than or equal to this value.

Parameters

'maxSideViaNeighbors <i>n_maxSideViaNeighbors</i>	The maximum number of neighbors allowed for a side via.
'maxSideViasWithNeighbors <i>n_maxSideViasWithNeighbors</i>	

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The maximum number of side vias that each have the maximum allowed neighbors as specified with `maxSideViaNeighbors`.

Examples

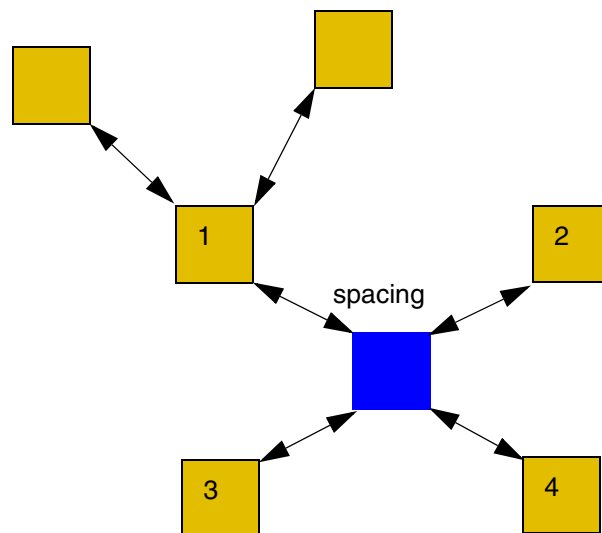
- Example 1: `minNestedViaSpacing` with `maxSideViaNeighbors`
- Example 2: `minNestedViaSpacing` with `maxSideViaNeighbors` and `maxSideViasWithNeighbors`

Example 1: `minNestedViaSpacing` with `maxSideViaNeighbors`

Consider the following constraint. The maximum number of neighbors allowed for a side via is 2. For the figure, let us assume that the spacing is as specified.

```
spacings (  
  (minNestedViaSpacing "Via1"  
    'maxSideViaNeighbors 2  
    0.1)  
) ;spacings
```

■ Side vias
■ Nested via



FAIL. The side via Via1 has 3 neighbors, including the nested via, which is more than the `maxSideViaNeighbors` value.

Virtuoso Technology Data Constraints Reference

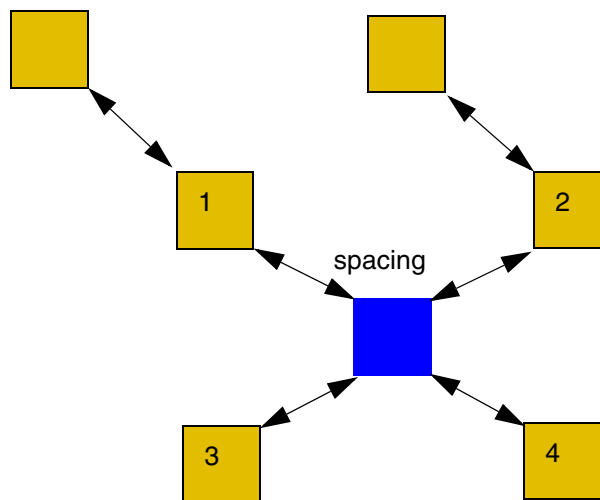
Via Construction Constraints

Example 2: *minNestedViaSpacing* with *maxSideViaNeighbors* and *maxSideViasWithNeighbors*

Consider the following constraint. The maximum number of neighbors allowed for a side via is 2. Further, only 1 side via should meet this requirement. For the figure, let us assume that the spacing is as specified.

```
spacings (  
  (minNestedViaSpacing "Via1"  
    'maxSideViaNeighbors 2  
    'maxSideViasWithNeighbors 1  
    0.1)  
); spacings
```

■ Side vias
■ Nested via



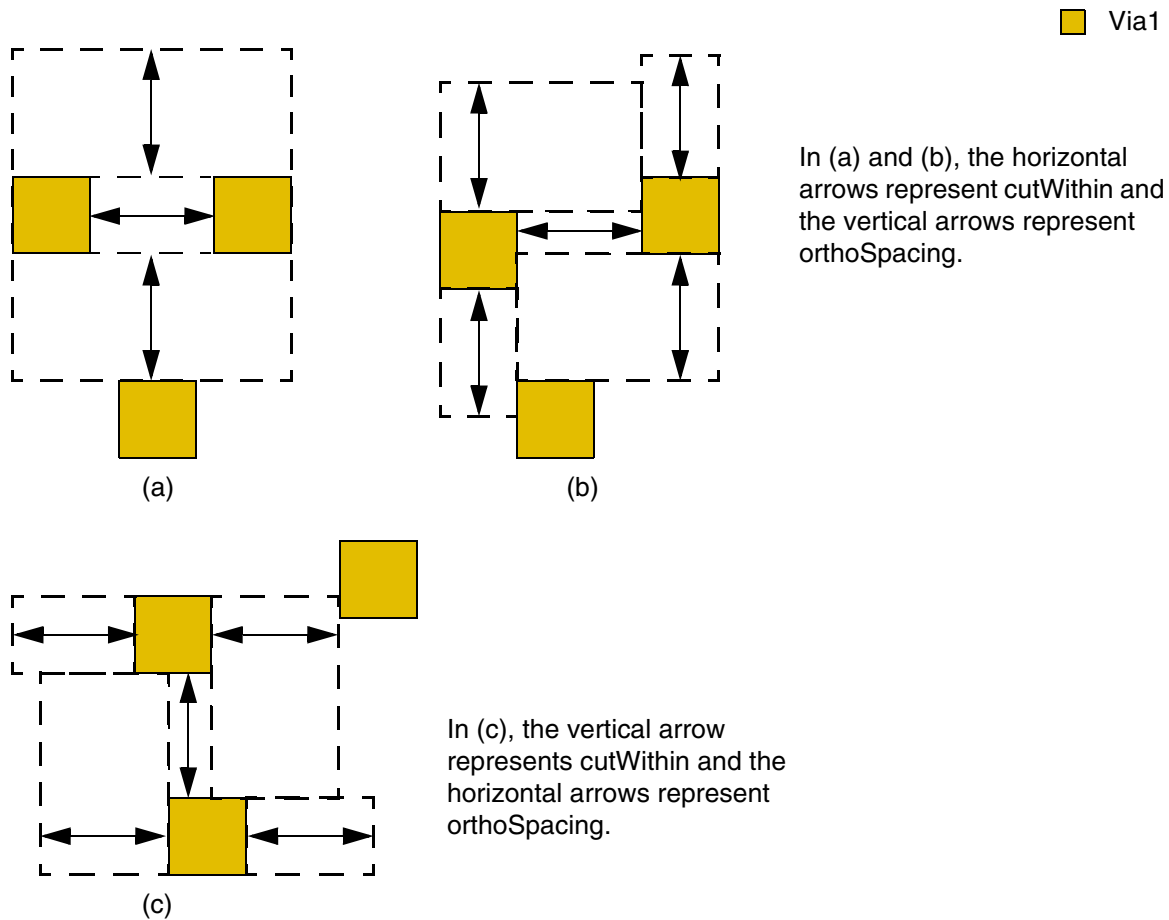
FAIL. In this case, Via1 only has 2 neighbors as required by `maxSideViaNeighbors`. However, the constraint is violated because both Via1 and Via2 have 2 neighbors each whereas `maxSideViasWithNeighbors` specifies that only one side via should meet the requirement of `maxSideViaNeighbors`.

minOrthogonalViaSpacing

```
spacingTables (
  ( minOrthogonalViaSpacing tx_cutLayer
    (( "distance" nil nil )
      [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum orthogonal spacing between an overlapping via cut pair and any other via cut based on the distance between the overlapping via cuts.

The following figure illustrates how orthogonal spacing is applied. If the distance between the overlapping via cuts is less than *cutWithin*, the spacing between the overlapping via cut pair and any other via cut must be greater than *orthoSpacing*.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"distance" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of a <i>table</i> row is as follows: <i>f_cutWithin f_orthoSpacing</i> where, <i>f_cutWithin</i> is the distance between the overlapping via cuts and <i>f_orthoSpacing</i> is the orthogonal spacing applied between an overlapping via cut pair and any other via cut when the distance between the overlapping via cuts is less than <i>f_cutWithin</i> . Type: A 1-D table specifying floating-point distance and spacing values.

Parameters

<i>f_default</i>	The spacing value to be used when no table entry applies.

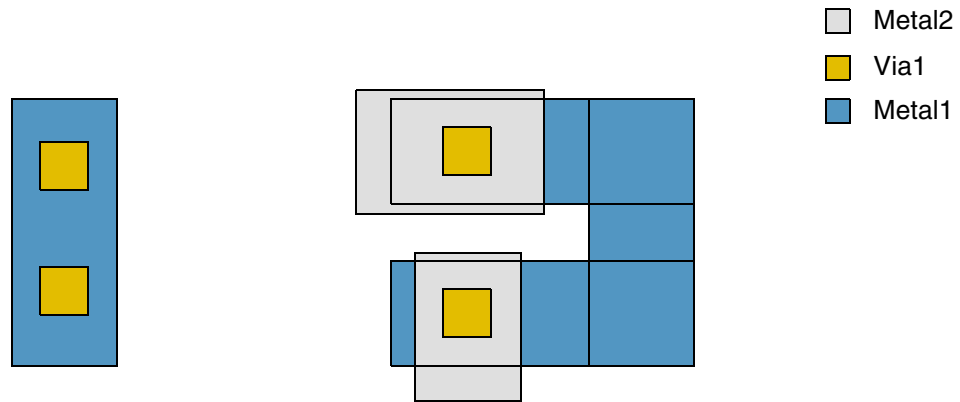
minParallelViaSpacing (One layer)

```

spacings (
    ( minParallelViaSpacing tx_layer
      ['centerToCenter]
      ['exceptSameNet | 'exceptSameMetal | 'exceptSameVia]
      f_spacing
    )
) ;spacings

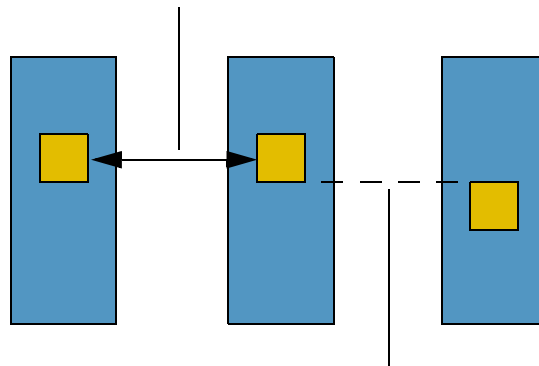
```

Specifies the minimum spacing between via cuts on a layer if the via cuts have parallel edges with an overlap greater than zero. The constraint does not apply if the via cuts share the same metal shape.



The constraint does not apply because the via cuts are on the same Metal1 shape.

The constraint applies because both conditions are satisfied: the via cuts are on different metal shapes and have an overlap greater than zero



The constraint does not apply because the via cuts have zero overlap.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_layer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between via cuts must be greater than or equal to this value.

Parameters

'centerToCenter	The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge.
'exceptSameNet 'exceptSameMetal 'exceptSameVia	The constraint does not apply to via cuts that are on the same net ('exceptSameNet), on a contiguous same-metal shape ('exceptSameMetal), or are overlapped by a single metal shape from above and by another single metal shape from below ('exceptSameVia).

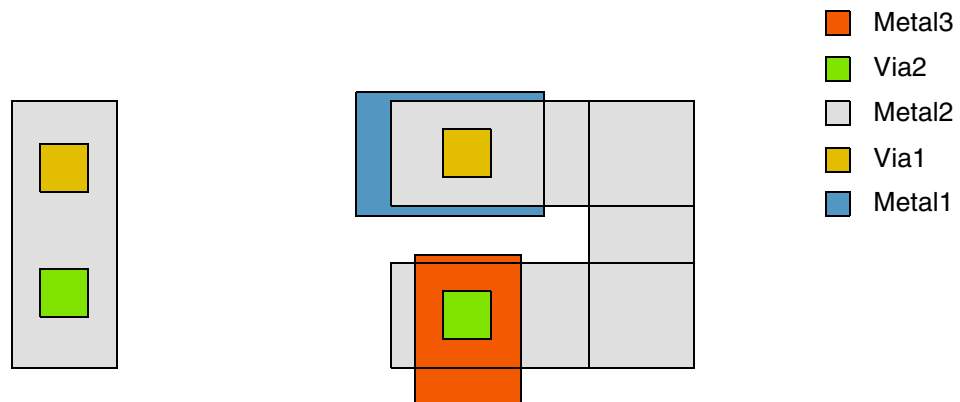
minParallelViaSpacing (Two layers plus metal)

```

spacings (
    ( minParallelViaSpacing tx_cutLayer1 tx_cutLayer2 tx_metalLayer
      f_spacing
    )
) ; spacings

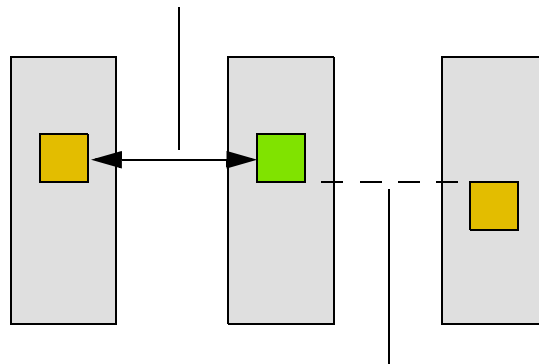
```

Specifies the minimum edge-to-edge spacing between via cuts on two different layers if the via cuts have parallel edges with an overlap greater than zero. The constraint does not apply if the via cuts share the same metal shape on the layer between the cut layers.



The constraint does not apply because the via cuts are on the same Metal2 shape.

The constraint applies because both conditions are satisfied: the via cuts are on different metal shapes and have an overlap greater than zero



The constraint does not apply because the via cuts have zero overlap.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_cutLayer1</i>	The first cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer2</i>	The second cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. The metal layer must be between the two cut layers. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing required between the via cuts.

Parameters

None

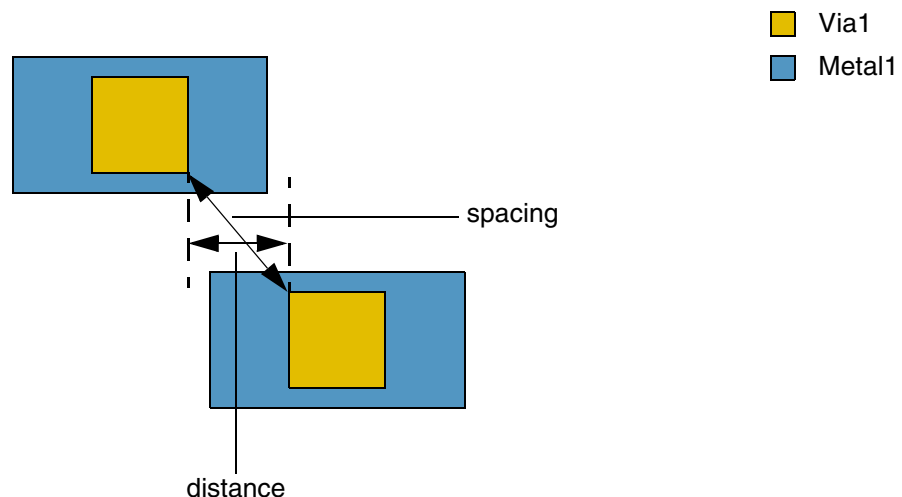
minParallelWithinViaSpacing

```

spacings (
  ( minParallelWithinViaSpacing tx_cutLayer
    ['centerToCenter]
    ['exceptSameNet]
    ['cutClass {f_width | (f_width f_length) | t_name}
      ['longEdgeOnly ['otherWidth]
        | ['otherExtension f_extension 'layer tx_metalLayer
          'paraLength f_parLength 'within f_within
        ]
      ]
    ]
  ]
  'cutDistance f_distance
  ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ]
  f_spacing
)
) ;spacings

```

Specifies the minimum spacing required between cut shapes if the distance between the cut shapes is less than *distance*.



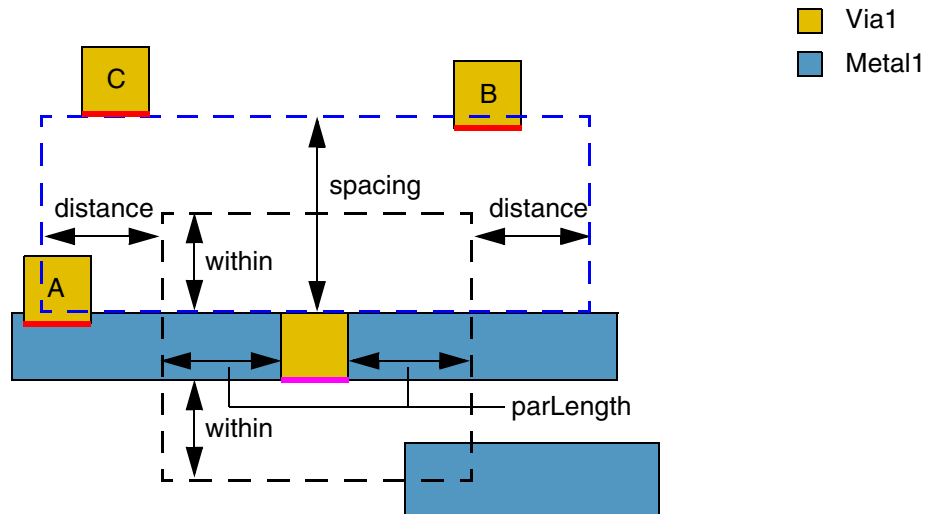
The constraint applies if the cut shapes are overlapped by a same-metal shape. However, if you do not want the constraint to apply between cut shapes on the same net, you can specify the 'exceptSameNet' parameter.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Optionally, the constraint applies only if a cut shape has a *metalLayer* extension less than *extension* on a side on which lies a neighboring *metalLayer* shape at a distance less than *within* from the cut shape, with parallel run length greater than or equal to *parLength* with the cut shape. If the specified parallel run length (*parLength*) is negative, the edge of the target cut shape is extended in both directions by the absolute value of the specified parallel run length and the constraint applies between the extended edge and any neighboring cut edge.

For example, in the figure below, the via cut has Metal1 extension less than *extension* on the bottom edge (shown in pink)—on the side on which lies a neighboring shape on layer Metal1 at a distance less than *within* from the via cut. The parallel run length between the via cut and the neighboring shape is *-parLength*. To prevent a violation, any red edge of a neighboring via cut that faces this via cut must lie outside the blue dotted window.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between cut shapes must be greater than or equal to this value.

Parameters

'centerToCenter	The spacing between cut shapes is measured center-to-center. Otherwise, the spacing is measured edge-to-edge. Type: Boolean
'exceptSameNet	The constraint does not apply between cut shapes on the same net. Type: Boolean
'cutClass { <i>f_width</i> (<i>f_width</i> <i>f_length</i>) <i>t_name</i> }	(ICADVM20.1 Only – 95512) The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint). ■ <i>f_width</i> : Width ■ <i>f_length</i> : Length ■ <i>t_name</i> : Name of the cut class
'longEdgeOnly	(ICADVM20.1 Only – 95512) The constraint applies only if a metal shape lies between the long edges of two rectangular cut shapes and is inside the common projected parallel run length of the cut shapes. Additionally, no cut shape of the same cut class should be present in the area between the long edges of the two rectangular cut shapes. Type: Boolean
'otherWidth	(ICADVM20.1 Only – 95511 and 95800) The constraint applies if the metal shape on the layer above the cut layer is within the common projected parallel run length of cut shapes and has width less than this value. This parameter is specified only if <code>longEdgeOnly</code> is specified.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
'otherExtension f_extension 'layer tx_metalLayer  
'paraLength f_paraLength 'within f_within
```

The constraint applies only if the cut shape has a *metalLayer* extension less than *extension* on a side on which lies a neighboring *metalLayer* shape at a distance less than *within* from the cut shape, with parallel run length greater than or equal to *paraLength* with the cut shape.

Spacing is checked between cut shapes only on the side opposite to that on which the neighboring shape is present. If the specified parallel run length (*paraLength*) is negative, the edge of the target cut shape is extended in both directions by the absolute value of the specified parallel run length, and the constraint applies between the extended edge and any neighboring cut edge.

```
'cutDistance f_distance
```

The constraint applies only if the distance between the cut shapes is less than this value.

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)  
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to only the specified purposes.

Type: String (purpose name)

Examples

- Example 1: minParallelWithinViaSpacing with cutClass and cutDistance
- Example 2: minParallelWithinViaSpacing with cutClass, longEdgeOnly, and cutDistance

Virtuoso Technology Data Constraints Reference

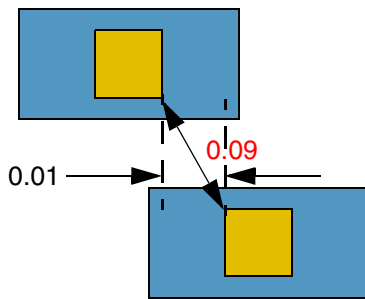
Via Construction Constraints

Example 1: *minParallelWithinViaSpacing* with *cutClass* and *cutDistance*

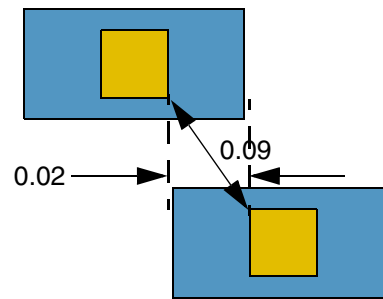
The spacing between the edges of Via1 via cuts must be at least 0.1 if the distance between two via cuts is less than 0.02.

```
spacings(  
  ( minParallelWithinViaSpacing "Via1"  
    'cutDistance 0.02  
    0.1  
  )  
) ;spacings
```

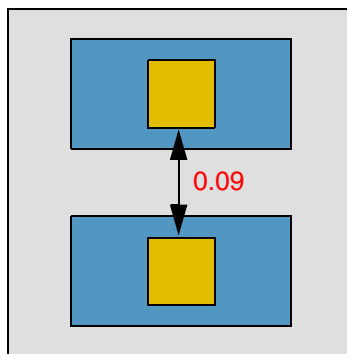
□ Metal2
■ Via1
■ Metal1



a) FAIL. The distance between the two via cuts is 0.01 (< 0.02), but the spacing between the via cuts is only 0.09 (< 0.1).



b) The constraint does not apply because the distance between the two via cuts is 0.02 (and not less than 0.02).



b) FAIL. The via cuts are overlapped by a same-metal shape, but the spacing between the via cuts is only 0.09 (< 0.1).

Virtuoso Technology Data Constraints Reference

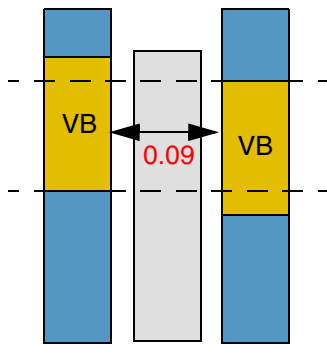
Via Construction Constraints

Example 2: *minParallelWithinViaSpacing* with *cutClass*, *longEdgeOnly*, and *cutDistance*

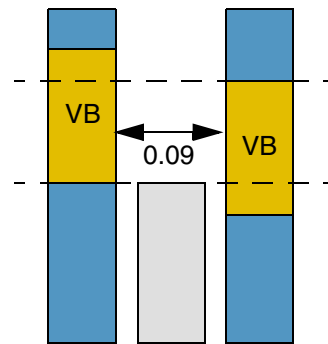
The spacing between the long edges of Via1 via cuts of type VB (0.4x0.6) must be at least 0.1 if the distance between two via cuts is less than 0.02.

```
spacings (
  ( minParallelWithinViaSpacing "Via1"
    'cutClass VB
    'longEdgeOnly
    'cutDistance 0.02
    0.1
  )
) ;spacings
```

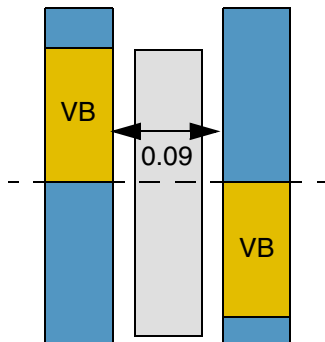
 Metal2
 Via1
 Metal1



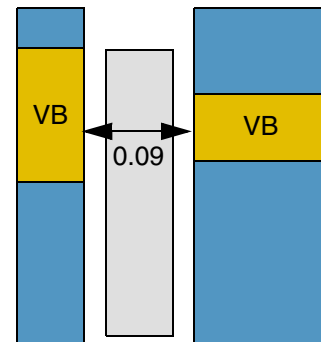
a) FAIL. The metal shape that lies between the via cuts is inside the common projected parallel run length of the via cuts, but the spacing between the via cuts is only 0.09 (<0.1).



b) The constraint does not apply because the metal shape that lies between the via cuts is outside the common projected parallel run length of the via cuts.



c) The constraint does not apply because the parallel run length between the long edges of the via cuts is zero.



d) The constraint does not apply because the spacing is measured between a long and a short edge. A violation would occur if 'longEdgeOnly' was not specified.

minSameMetalSharedEdgeViaSpacing

```

spacings (
    ( minSameMetalSharedEdgeViaSpacing tx_cutLayer
      'within f_within
      ['above]
      ['cutClass { f_width | (f_width f_length) | t_name }]
      ['exceptTwoEdges]
      ['numCuts x_numCuts]
      f_spacing
    )
) ;spacings

```

Specifies the minimum spacing between two via cuts with a parallel overlap greater than zero and with neighboring wires at a distance less than *within*, overlapped by a single metal shape that covers the entire length of the common projection between them from above and/or below.

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing between the via cuts.

Parameters

'within <i>f_within</i>	The neighboring wires must be present at a distance less than this value from the via cuts.
'above	The constraint applies only if the via cuts are overlapped by a same-metal shape from above. Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'cutClass { *f_width* | (*f_width* *f_length*) | *t_name* }

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

'exceptTwoEdges

The constraint does not apply if at least one via cut has neighboring wires on opposite edges at a distance less than *within* from it.

Type: Boolean

'numCuts *x_numCuts*

The constraint does not apply if there exist at least these many via cuts overlapped by the a single metal shape on the layers above and below.

Examples

- [Example 1: minSameMetalSharedEdgeViaSpacing with within and exceptTwoEdges](#)
- [Example 2: minSameMetalSharedEdgeViaSpacing with within](#)

Virtuoso Technology Data Constraints Reference

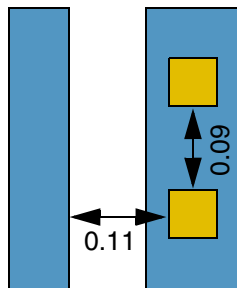
Via Construction Constraints

Example 1: *minSameMetalSharedEdgeViaSpacing* with *within* and *exceptTwoEdges*

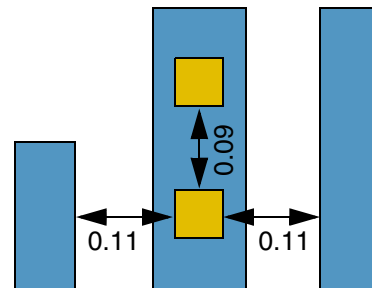
Two via cuts with a parallel overlap greater than zero and neighboring wires at a distance less than 0.12 on the same edge must have a spacing greater than or equal to 0.1.

```
spacings (  
  ( minSameMetalSharedEdgeViaSpacing "Via1"  
    'within 0.12  
    'exceptTwoEdges  
    0.1  
  )  
) ;spacings
```

■ Via1
■ Metal1



a) FAIL. The via cuts have a parallel overlap greater than zero and a neighboring wire at a distance 0.11 (<0.12), but the spacing between the via cuts is only 0.09 (<0.1).



b) The constraint does not apply because at least one via cut has neighboring wires at a distance 0.11 (<0.12) on opposite edges.

Virtuoso Technology Data Constraints Reference

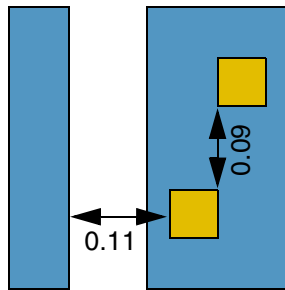
Via Construction Constraints

Example 2: *minSameMetalSharedEdgeViaSpacing* with *within*

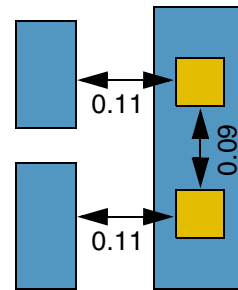
Two via cuts with a parallel overlap greater than zero and neighboring wires at a distance less than 0.12 must have a spacing greater than or equal to 0.1.

```
spacings (
  ( minSameMetalSharedEdgeViaSpacing "Via1"
    'within 0.12
    0.1
  )
) ;spacings
```

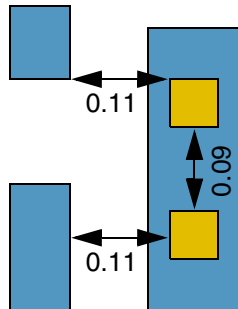
Metal2
 Via1
 Metal1



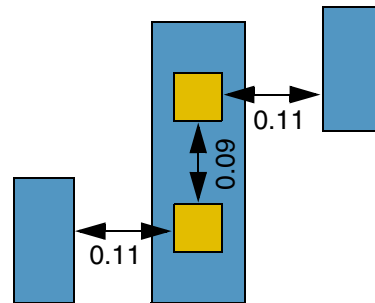
a) The constraint does not apply because the via cuts do not have a parallel overlap greater than zero.



b) FAIL. The neighbors can be different wires. However, the spacing between the via cuts is only 0.09 (<0.1).



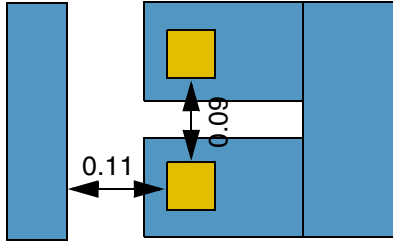
c) The constraint does not apply because the top via cut does not have a neighboring wire.



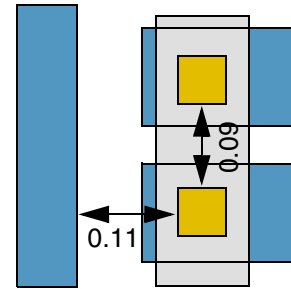
d) FAIL. Each via cut has a neighboring wire at a distance 0.11 (<0.12) from it. However, the spacing between the via cuts is only 0.09 (<0.1).

Virtuoso Technology Data Constraints Reference

Via Construction Constraints



e) The constraint does not apply because the via cuts do not share a common metal shape that covers the entire projection between them.



f) FAIL. The via cuts share a common metal shape that covers the entire projection between them from above. However, the spacing between the via cuts is only 0.09 (< 0.1).

minViaGroupSpacing

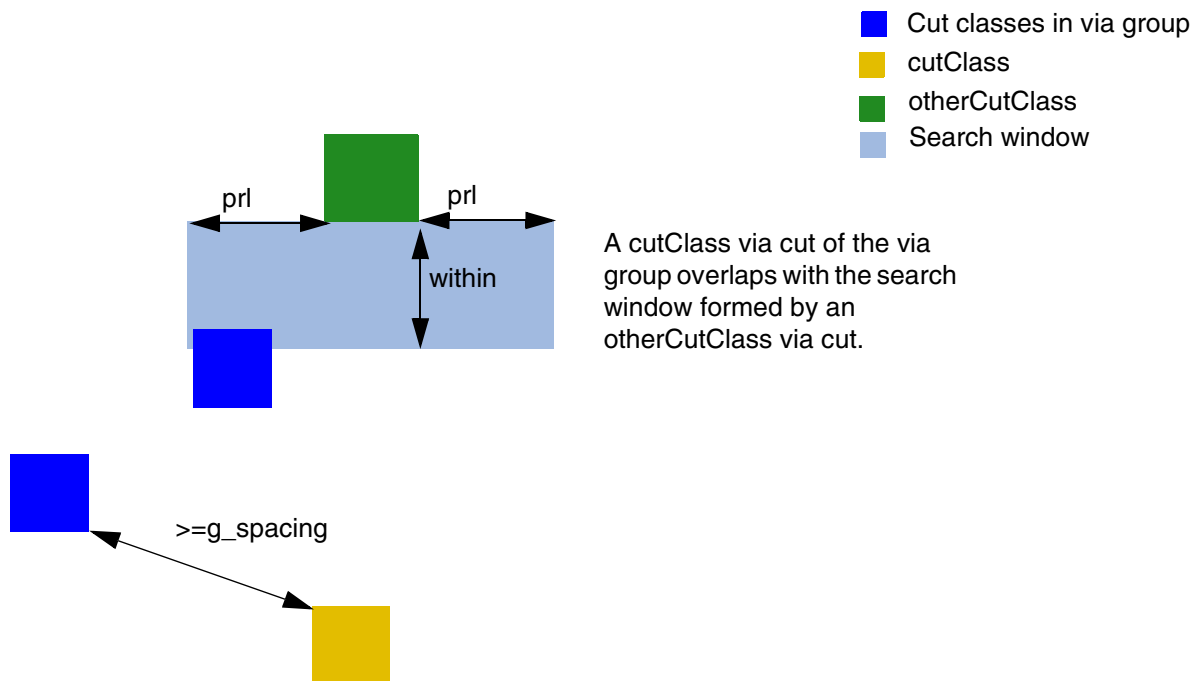
```

spacings (
  ( minViaGroupSpacing tx_layer
    'groupName t_groupName
    ['cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}]
    'numCuts x_numCuts
    ['prl f_prl
    'within f_within
    'otherCutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}]
    ['sameMask | 'diffMask]
    ['vertical | 'horizontal ['cutPrl g_cutPrl]]
    ['twoSides]
    ['groupToGroup ['noMergedCuts]]
    ['mergedCuts]
    ['otherLayer g_otherLayer]
    g_spacing)
) ;spacings

```

(ICADVM20.1 Only – 95511) Specifies the spacing of a cut in a via group to a cut of a given cut class.

The constraint applies selectively for a specified via group only when a cut of the specified cutClass is a neighboring cut of the top or bottom edge of the specified otherCutClass in the specified within distance when the parallel run length or prl is greater than the given value.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_layer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>g_spacing</i>	The spacing between the cut shapes must be greater than or equal to this value.

Parameters

`'groupName t_groupName`

The constraint applies to the specified via group.

`'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}`

The constraint applies to the cut class specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

`'numCuts x_numCuts`

The constraint applies only if the number of via cuts in the group is exactly this value.

`'prl f_prl`

This parameter, with `within`, forms the search window for `otherCutClass`.

Type: Floating-point values specifying a range of parallel run lengths

`'within f_within`

This parameter, with `prl`, forms the search window for `otherCutClass`.

`'otherCutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The constraint applies only if any cut of the via group specified by `groupName` overlaps the search window of a cut of this cut class, which is specified by width, by width and length, or by name (as defined in a `cutClasses` constraint).

- `f_cutWidth`: Width
- `f_cutLength`: Length
- `t_name`: Name of the cut class

`'sameMask | 'diffMask`

The mask type. If mask type is not specified, the constraint applies to all cuts on the layer.

- `'sameMask`: The constraint applies only on cuts with the same mask.
- `'diffMask`: (ICADVM20.1 Only – 95511 and 95800) The constraint applies only to via cuts on different masks.

Type: Boolean

`'horizontal | 'vertical [ 'cutPr1 g_cutPr1]`

The constraint applies only in the given direction on spacing between two cuts with a parallel run length more than 0 or more than the specified `cutPr1` value.

`'twoSides` (ICADVM20.1 Only – 95511 and 95800) The constraint applies only when there are neighbor cuts on both sides.

`'groupToGroup [ 'noMergedCuts]`

The constraint applies only on cuts of different via groups. Cuts of the same group are merged before the spacing is applied.

(ICADVM20.1 Only – 95511 and 95800) `noMergedCuts` specifies that cuts should not be merged before applying the spacing.

`'mergedCuts` (ICADVM20.1 Only – 95511 and 95800) The cut spacing is applied on the bounding box formed around cuts of a group.

`'otherLayer g_otherLayer`

(ICADVM20.1 Only – 95511 and 95800) The constraint considers the inter-layer cut spacing between a via group on the current layer to a cut on an adjacent cut layer.

Virtuoso Technology Data Constraints Reference

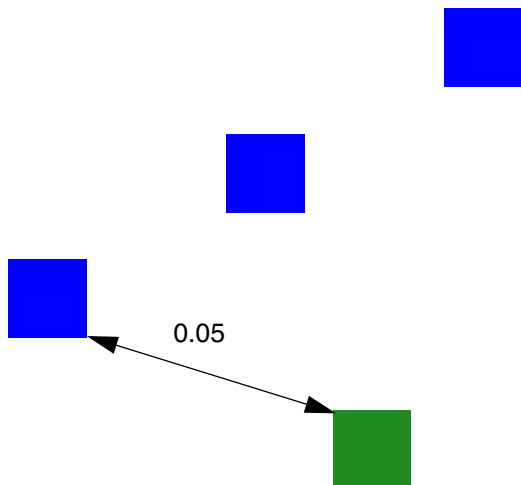
Via Construction Constraints

Example

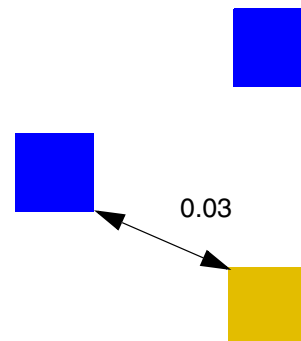
Cuts of the VA cutClass need to have a minimum spacing of 0.05 to the via group VGA if the group has at least 3 members. Cuts of the VB cutClass need to have a minimum spacing of 0.03 to VGA if the group has at least 2 members.

```
spacings(  
  (minViaGroupSpacing "Via1"  
    'groupName "VGA" 'cutClass "VA" 'numCuts 3  
    0.05)  
  (minViaGroupSpacing "Via1"  
    'groupName "VGA" 'cutClass "VB" 'numCuts 2  
    0.03)  
) ;spacings
```

■ Cut classes in VGA
■ cutClass VA
■ cutClass VB



PASS. The required spacing of 0.05 is met.



PASS. The required spacing of 0.03 is met.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

minViaSpacing (One layer)

```
spacings (
  ( minViaSpacing tx_cutLayer
    ['shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
      | 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide
    ]
    ['centerToCenter]
    ['horizontal | 'vertical]
    ['sameNet | 'sameMetal | 'sameVia]
    ['viasOnSameNet | 'viasOnSameMetal | 'viasOnSameVia]
    ['exceptSameNet | 'exceptSameMetal]
    ['exceptViasOnSameNet | 'exceptViasOnSameMetal | 'exceptViasOnSameVia]
    ['sameMask | 'diffMask]
    ['area f_area]
    ['overLayer tx_layer
      ['overLayerWidth f_layerWidth]
      ['exceptGap f_gap]
      ['exceptSameMetalOverlap]
    ]
    ['length f_length | 'exactLength f_length ['exactSpacing]
      | 'prlRange f_prlRange]
    [
      ['cutClass { f_width | ( f_width f_length ) | t_name }
        ['otherCutClass { f_width | ( f_width f_length ) | t_name }]]
      |
      ['fixedCutClassList l_fixedCutClassNames
        ['otherFixedCutClassList l_otherFixedCutClassNames]]
    ]
    ['enclosingLayer tx_enclosingLayer
      ['viaEdgeType x_edgeType
        ['bothCuts | 'otherViaEdgeType x_edgeType2]
        ['sizeByTouchedCorners (f_sizeBy1 f_sizeBy2)
          ['includedEdgeAligned
            ['alignedHorizontal|'alignedVertical]]]
        ['orthogonalEdgePair
          ['orthogonalEnclosingMetal]]
      ]
    ]
    ['sizeBy f_sizeBy]
    ['exceptExactAligned]
    ['exceptRanges l_exceptRanges]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      |
      ['outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    [ 'ignoreRegion ] | [ 'abutRegion ]
    ['facingEdges]
    ['deltaVoltage f_voltage]
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
['horizontalSpacingRange f_spacingRange]
['verticalSpacingRange f_spacingRange]
['farEnds]
f_spacing ['manhattan]
)
) ;spacings

spacingTables(
  ( minViaSpacing tx_cutLayer
    (( "width" nil nil ["width" nil nil] )
      ['shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
        | 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide
      ]
      ['centerToCenter]
      ['horizontal | 'vertical]
      ['sameNet | 'sameMetal | 'sameVia]
      ['viasOnSameNet | 'viasOnSameMetal | 'viasOnSameVia]
      ['exceptSameNet | 'exceptSameMetal]
      ['exceptViasOnSameNet | 'exceptViasOnSameMetal | 'exceptViasOnSameVia]
      ['sameMask | 'diffMask]
      ['area f_area]
      ['overLayer tx_layer
        ['overLayerWidth f_layerWidth]
        ['exceptGap f_gap]
        ['exceptSameMetalOverlap]
      ]
      ['length f_length | 'exactLength f_exactLength ['exactSpacing]
        'prlRange f_prlRange]
      [
        ['cutClass { f_width | ( f_width f_length ) | t_name }
          ['otherCutClass { f_width | ( f_width f_length ) | t_name }]]
        |
        ['fixedCutClassList l_fixedCutClassNames
          ['otherFixedCutClassList l_otherFixedCutClassNames]]
      ]
      ['enclosingLayer tx_enclosingLayer
        ['viaEdgeType x_edgeType
          ['bothCuts | 'otherViaEdgeType x_edgeType2]
          ['sizeByTouchedCorners (f_sizeBy1 f_sizeBy2)
            ['includedEdgeAligned
              ['alignedHorizontal|'alignedVertical]]]
          ['orthogonalEdgePair
            ['orthogonalEnclosingMetal]]
        ]
      ]
      ['sizeBy f_sizeBy]
      ['exceptExactAligned]
      ['above | 'below | 'bothAboveBelow]
      ['exceptRanges l_exceptRanges]
      ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    )
  )
)
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
    |
    'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ]
  [ 'ignoreRegion ] | [ 'abutRegion ]
  ['facingEdges]
  ['deltaVoltage f_voltage]
  ['horizontalSpacingRange f_spacingRange]
  ['verticalSpacingRange f_spacingRange]
  ['farEnds]
  [f_default]
)
(g_table) ['manhattan]
)
) ;spacingTables
```

Specifies the minimum spacing between via cuts that are on the same net, metal, or via. The required spacing can be independent of the width of the enclosing metal shape or can be dependent on the width of the enclosing metal shapes on the layers above or below, or both. If the enclosing metal shape is rectilinear, the width of the narrowest portion of the shape is considered.

Optional parameters determine whether the spacing is measured center-to-center or edge-to-edge, whether the constraint applies to a certain connectivity type, and whether the constraint applies to via cuts that have an area larger than the specified value. In some processes, the constraint applies only if the via cuts have a parallel run length greater than or equal to the specified value or exactly equal to the specified value, and, in some processes, the constraint does not apply to via cuts that are exactly aligned. You can also use this constraint to define the spacing for a cut class.

Values

<i>tx_cutLayer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing independent of width.
"width" nil nil	This identifies the index for <i>table</i> .

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

g_table

The format of the 1-D table is as follows:

(f_index f_spacing)

where, *f_spacing* is the minimum spacing required between the via cuts when the width of the metal shape enclosing the via cuts is greater than or equal to *f_index*. By default, the required spacing is determined based on the width of the wider of the enclosing metal shapes on the layers above and below.

The format of the 2-D table is as follows:

(f_index1 f_index2 f_spacing)

where, *f_index1* and *f_index2* represent the widths of the enclosing shapes on the layers above and below.

Type: A 1-D or 2-D table specifying floating-point width and spacing values.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

'shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
| 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide

(ICADVM20.1 Only – 95511) The sides between which spacing is measured.

- 'shortSideToShortSide: Spacing is measured between a short side of one shape and a short side of the other shape.
- 'shortSideToLongSide: Spacing is measured between a short side of one shape and a long side of the other shape.
- 'shortSideToAnySide: Spacing is measured between a short side of one shape and any side of the other shape.
- 'longSideToLongSide: Spacing is measured between a long side of one shape and a long side of the other shape.
- 'longSideToAnySide: Spacing is measured between a long side of one shape and any side of the other shape.
- 'anySideToAnySide: Spacing is measured between any side of one shape and any side of the other shape.

Type: Boolean

'centerToCenter

The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge.

Type: Boolean

'horizontal | 'vertical

(ICADVM20.1 Only – 95512) If a spacing direction is specified for the constraint, then the parallel run length is measured in the orthogonal direction. If direction is not specified, the constraint applies in any direction.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'sameNet | 'sameMetal | 'sameVia

The connectivity type. If the connectivity type is not specified, the constraint applies to any connectivity type, including any two cuts on the same cut layer with no common metal or net.

- 'sameNet: The constraint applies broadly to all via cuts that are on the same net.
- 'sameMetal: The constraint applies to a subset of via cuts that meet the sameNet condition and are on contiguous same-metal shapes.
- 'sameVia: The constraint applies to a smaller subset of via cuts that meet the sameMetal condition and are overlapped by a single metal shape from above and by another single metal shape from below.

This parameter may be specified only if another connectivity type (including `viasOnSame*`, `exceptSame*`, and `viasOnSame*`) is not specified.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'viasOnSameNet | 'viasOnSameMetal | 'viasOnSameVia

(ICADVM20.1 Only – 95511) The constraint applies selectively to vias of the specified connectivity type. If none of these connectivity types are specified, the constraint applies to any connectivity type, including any two cuts on the same cut layer with no common metal or net.

- 'viasOnSameNet: The constraint applies broadly to all cut shapes that are either logically assigned the same net or electrically connected to the same net.



- 'viasOnSameMetal: The constraint applies to a subset of cut shapes that meet the `viasOnSameNet` condition and are also electrically connected by a contiguous metal on either the layer below or the layer above.



- 'viasOnSameVia: The constraint applies to a smaller subset of cut shapes that meet the `viasOnSameMetal` condition and are also electrically connected by a contiguous metal on both the layer below and the layer above.



This parameter may be specified only if another connectivity type (including `sameNet`, `sameMetal`, `sameVia`, `exceptSameNet`, `exceptSameMetal`, and `exceptViasOnSame*`) is not specified.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'exceptSameNet | 'exceptSameMetal

(ICADVM20.1 Only – 95512) The constraint does not apply to via cuts with the specified connectivity type. If neither of these connectivity types are specified, the constraint applies to any connectivity type, including any two cuts on the same cut layer with no common metal or net.

- 'exceptSameNet: The constraint does not apply to via cuts on the same net.
- 'exceptSameMetal: The constraint does not apply to a subset of via cuts that meet the `exceptSameNet` condition and are enclosed by a contiguous same-metal shape.

This parameter may be specified only if another connectivity type (including `sameNet`, `sameMetal`, `sameVia`, `viasOnSame*`, and `exceptViasOnSame*`) is not specified.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'exceptViasOnSameNet` | `'exceptViasOnSameMetal` |
`'exceptViasOnSameVia`

(ICADVM20.1 Only – 95511) The constraint does not apply to vias of the specified connectivity type. If none of these connectivity types are specified, the constraint applies to any connectivity type, including any two cuts on the same cut layer with no common metal or net.

- `'exceptViasOnSameNet`: The constraint does not apply to a broad set of via cuts that are either logically assigned the same net or electrically connected to the same net.
- `'exceptViasOnSameMetal`: The constraint does not apply to a subset of via cuts that meet the `exceptViasOnSameNet` condition and are also electrically connected by a contiguous metal on either the layer below or the layer above.
- `'exceptViasOnSameVia`: The constraint does not apply to a smaller subset of via cuts that meet the `exceptViasOnSameMetal` condition and are also electrically connected by a contiguous metal on both the layer below and the layer above.

This parameter may be specified only if another connectivity type (including `sameNet`, `sameMetal`, `sameVia`, `exceptSameNet`, `exceptSameMetal`, and `viasOnSame*`) is not specified.

`'sameMask` | `'diffMask`

The mask type. If mask type is not specified, the constraint applies to all cuts on the layer.

- `'sameMask`: (ICADVM20.1 Only – 95512) The constraint applies only to via cuts on the same mask.
- `'diffMask`: (ICADVM20.1 Only – 95511) The constraint applies only to via cuts on different masks.

Type: Boolean

`'area f_area`

The constraint applies only if the area of a via cut is greater than this value.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'overLayer tx_layer`

(ICADVM20.1 Only – 95512) The constraint applies only if both via cuts overlap a single shape on this layer. If `exceptGap` or `exceptSameMetalOverlap` is specified, this condition is not checked.

Type: String (layer name) or Integer (layer number)

`'overLayerWidth f_layerWidth`

(ICADVM20.1 Only – 95512) The constraint applies only if the width of the overlapping shape is less than or equal to this value.

`'exceptGap f_gap`

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply if there is a gap of exactly this value between the via cuts that overlap the shape on the `overLayer` layer.

`'exceptSameMetalOverlap`

(Virtuoso Layout Suite EXL and higher tiers) The constraint does not apply if the via cuts that overlap the shape on the `overLayer` layer are on the same metal.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'length *f_length*

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 `minViaSpacing` constraints are defined in an AND constraint group, the constraint is applied by evaluating the *length* values, specified in the ascending order, to locate the first *length* value for which the actual parallel run length 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 parallel run length values specified as -1.0, -0.05, 0, 0.05, 1, the constraint value that applies is determined as follows:

Actual <i>length</i> (prl)	Use constraint value for <i>length</i>
<-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

'exactLength *f_exactLength*

The constraint applies only if the parallel run length between the two via cuts is equal to this value.

'exactSpacing

The spacing between the via cuts must be equal to the constraint value.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'prlRange f_prlRange`

(ICADVM20.1 Only – 95511) The constraint applies only when the parallel run length between two shapes is in this range. The parallel run length is measured orthogonal to the direction parameter, when it is specified.

`'cutClass { f_width | (f_width f_length) | t_name }`

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint). The constraint applies to edges specified by `x_edgeType`.

`'otherCutClass { f_width | (f_width f_length) | t_name }`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if one cut class is of the type `cutClass` and the other cut class is of the type `otherCutClass` and are of the type specified by `x_edgeType2`.

`'fixedCutClassList l_fixedCutClassNames`

(Virtuoso Layout Suite EXL and higher tiers) The constraint provides the list of cut classes for which this constraint specifies the spacing.

`'otherFixedCutClassList l_otherFixedCutClassNames`

(Virtuoso Layout Suite EXL and higher tiers) The constraint provides the list of cut classes. A cut in `fixedCutClassList` will have specified spacing value with any cut in `otherFixedCutClassList`.

`'enclosingLayer tx_enclosingLayer`

(ICADVM20.1 Only – 95512) The enclosing layer for `'viaEdgeType` extension checks.

Type: String (layer name) or Integer (layer number)

`'viaEdgeType x_edgeType`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to the via cut edges of this type (the type is defined in the `viaEdgeType` constraint).

`'bothCuts`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only if both via cuts satisfy `viaEdgeType`.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'otherViaEdgeType x_edgeType2`

(Virtuoso Layout Suite EXL and higher tiers) (Virtuoso Layout Suite EXL and higher tiers) The constraint applies between via edges where one via edge is of the type `viaEdgeType` and the other via edge is of the type `otherViaEdgeType`. The type is defined in the `viaEdgeType` constraint.

`'sizeByTouchedCorners (f_sizeBy1 f_sizeBy2)`
`['includedEdgeAligned ['alignedHorizontal|'alignedVertical]]`

(ICADVM20.1 Only – 95511) The constraint is applied after expanding by `sizeBy1` the pair of opposite edges that satisfy `'viaEdgeType` and by `sizeBy2` the other pair of opposite edges of two via cuts only if the two expanded via cuts touch at the corners.

The `includedEdgeAligned` parameter indicates that the `sizeByTouchedCorners` spacing is applied if the edges of the expanded window are aligned. The alignment is considered to be in any direction, unless either `'alignedHorizontal` or `'alignedVertical` is specified.

`'orthogonalEdgePair` (Virtuoso Advanced Node for Layout Standard) The constraint checks for orthogonal pairs of edges when `centerToCenter` is `false` and `viaEdgeType` is defined.

Type: Boolean

`'orthogonalEnclosingMetal`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only to orthogonal enclosing metal shapes.

`'sizeBy f_sizeBy`

(Virtuoso Advanced Node for Layout Standard) The constraint is applied after expanding all via cut edges by this value.

If both `'enclosingLayer` and `'sizeBy` are specified, the spacing is measured between the AND of the enclosing layer extension and the via extended by the `sizeBy` value. 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.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'exceptExactAligned`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to those via cuts that are not exactly aligned.

Type: Boolean

`'above` | `'below` | `'bothAboveBelow`

(ICADVM20.1 Only – 95511) The metal shape that is used to determine the spacing between the via cuts. By default, the required spacing is determined based on the width of the wider of the two metal shapes. Only those metal shapes that overlap both via cuts are considered.

- `'above`: The spacing is determined based on the width of the metal shape on the layer above.
- `'below`: The spacing is determined based on the width of the metal shape on the layer below.
- `'bothAboveBelow`: The spacing is determined based on the width of the narrower of the two metal shapes on the layers above and below.

`'exceptRanges` *`l_exceptRanges`*

The constraint does not apply if the spacing between via cuts is in this range.

Type: A list of floating-point values specifying a range of spacings that are exempted.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)`
`| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)`

The presence or absence of one or more specified layers determines if the constraint applies.

- `'insideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- `'outsideLayers`: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to the specified purposes.

Type: String (purpose name)

`'ignoreRegion` (Virtuoso Layout Suite EXL and higher tiers) The constraint applies if the shapes on the `insideLayers` intersect any via shape but do not intersect the region between the shapes.

`'abutRegion` (Virtuoso Layout Suite EXL and higher tiers) The constraint applies if the shapes on the `insideLayers` abut the region between the shapes but do not overlap the region.

`'deltaVoltage f_voltage`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if both the cut shape and the metal shape have voltages and the delta voltage between the *layer1* shape and the *layer2* shape is greater than this value.

The delta voltage is calculated as follows:

- For asynchronous nets:

$$\max(V1_{\max}, V2_{\max}) - \min(V1_{\min}, V2_{\min})$$

- For synchronous nets:

$$\max(|V1_{\max} - V2_{\max}|, |V1_{\min} - V2_{\min}|)$$

'facingEdges

(Virtuoso Advanced Node for Layout Standard) The constraint applies only when there is no obstruction between edges.

Type: Boolean

'horizontalSpacingRange *f_spacingRange*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when the horizontal spacing between two shapes is in the specified range.

'verticalSpacingRange *f_spacingRange*

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only when the vertical spacing between two shapes is in the specified range.

'farEnds

(Virtuoso Layout Suite EXL and higher tiers) If specified, spacing is calculated between the far ends, including edges, of the vias and the constraint is applicable only for positive parallel run lengths between shapes.

f_default

The spacing value to be used when no table entry applies.

'manhattan

The constraint uses Manhattan distance, which allows a larger spacing at the corners. By default, the constraint uses Euclidean measurement. See [Euclidean and Manhattan Spacing Constraints](#).

Type: Boolean

Examples

- [Example 1: minViaSpacing with overLayer and overLayerWidth](#)
- [Example 2: minViaSpacing with centerToCenter, enclosingLayer, and viaEdgeType](#)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

- Example 3: minViaSpacing with exactLength, enclosingLayer, viaEdgeType, bothCuts, and sizeBy
- Example 4: minViaSpacing with fixedCutClassList and otherFixedCutClassList
- Example 5: minViaSpacing with exceptExactAligned
- Example 6: minViaSpacing with centerToCenter, cutClass, enclosingLayer, and sizeByTouchedCorners
- Example 7: minViaSpacing with orthogonalEdgePair
- Example 8: minViaSpacing with orthogonalEdgePair and orthogonalEnclosingMetal
- Example 9: minViaSpacing with exceptRanges
- Example 10: minViaSpacing with exceptRanges and facingEdges
- Example 11: minViaSpacing with exceptGap and exceptSameMetalOverlap
- Example 12: minViaSpacing with type of spacing
- Example 13: minViaSpacing with overLayer
- Example 14: minViaSpacing with verticalSpacingRange and horizontalSpacingRange
- Example 15: minViaSpacing with farEnds

Virtuoso Technology Data Constraints Reference

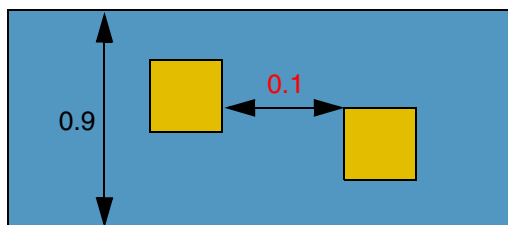
Via Construction Constraints

Example 1: minViaSpacing with overLayer and overLayerWidth

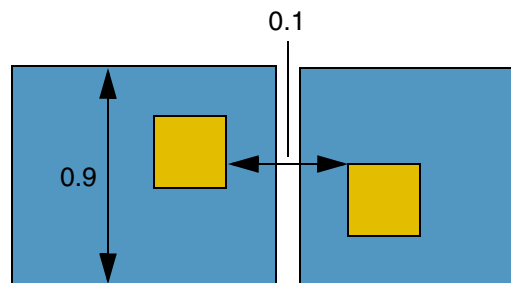
The spacing between two via cuts must be at least 0.2 if they overlap a Metal1 shape with width less than or equal to 0.09.

```
spacings(  
  ( minViaSpacing "Via1"  
    'overLayer "Metal1" 'overLayerWidth 0.9  
    0.2  
  )  
) ;spacings
```

■ Via1
■ Metal1



a) FAIL. Both via cuts overlap the same Metal1 shape with width 0.09. However, the spacing between these via cuts is 0.1 (< 0.2).



b) The constraint does not apply because the two via cuts do not overlap the same Metal1 shape.

Virtuoso Technology Data Constraints Reference

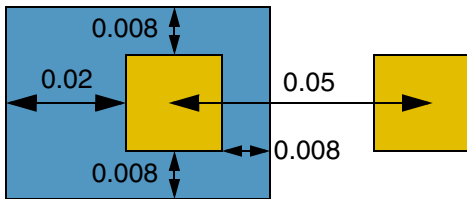
Via Construction Constraints

Example 2: *minViaSpacing* with *centerToCenter*, *enclosingLayer*, and *viaEdgeType*

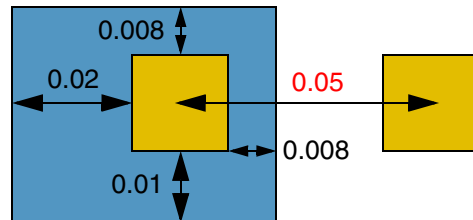
The center-to-center spacing between two via cuts must be at least 0.06. The constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.009.

```
spacings (
  ( viaEdgeType "Via1"
    'anyOppositeExtension 0.009
    'negateAnyOppositeExtension
      1
  )
  ( minViaSpacing "Via1"
    'centerToCenter
    'enclosingLayer "Metal1"
    'viaEdgeType 1
    0.06
  )
) ;spacings
```

■ Via1
■ Metal1



a) The constraint does not apply because the left via cut has a pair of opposite edges with 0.008 Metal1 extension (< 0.009), which means that the via cut edges are not of type 1.



b) FAIL. There are no opposite edges with Metal1 extension less than or equal to 0.009, but the spacing between the two cut vias is 0.05 (< 0.06).

Virtuoso Technology Data Constraints Reference

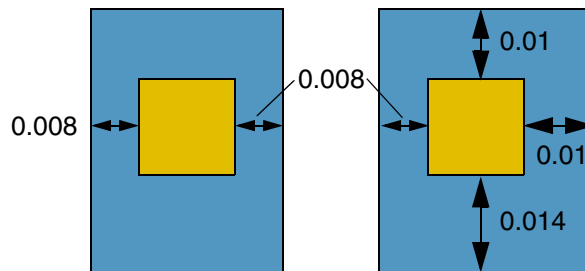
Via Construction Constraints

Example 3: *minViaSpacing* with *exactLength*, *enclosingLayer*, *viaEdgeType*, *bothCuts*, and *sizeBy*

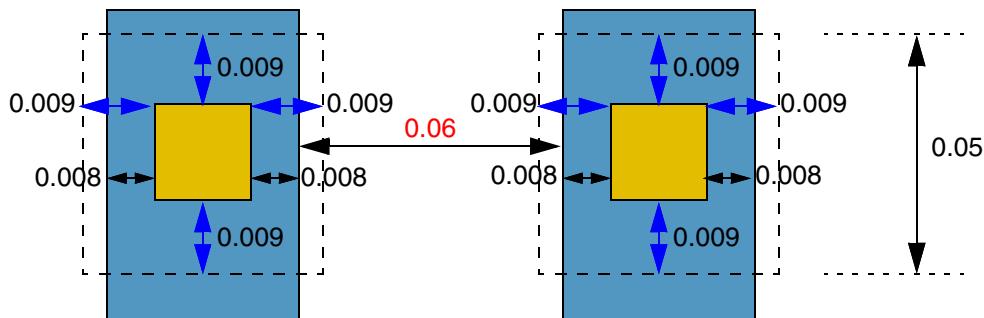
The spacing between two via cuts expanded by 0.009 must be at least 0.07 if both via cuts have Metal1 extensions less than or equal to 0.009 on a pair of opposite edges and the parallel run length between them is 0.05.

```
spacings(
  ( viaEdgeType "Via1"
    'anyOppositeExtension 0.009
    2
  )
  ( minViaSpacing "Via1"
    'exactLength 0.05
    'enclosingLayer "Metal1"
    'viaEdgeType 2 'bothCuts
    'sizeBy 0.009
    0.07
  )
);spacings
```

■ Via1
■ Metal1



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, 'viaEdgeType' 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).

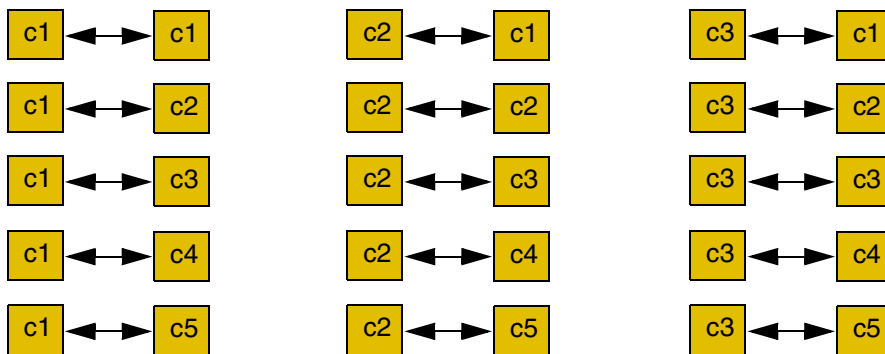
Example 4: minViaSpacing with fixedCutClassList and otherFixedCutClassList

Consider the following scenarios that illustrate the use of the `fixedCutClassList` and `otherFixedCutClassList` parameters.

In the following case, the spacings between all combinations of the specified cut classes must be ≥ 0.02 .

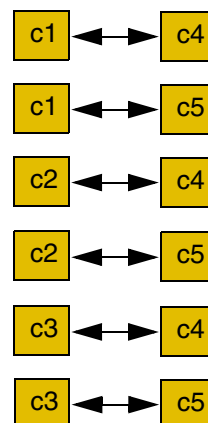
```
spacings(
  ( minViaSpacing "Via1"
    'fixedCutClassList ("c1" "c2" "c3")
    0.02
  )
);spacings
```

■ Via1



In this case, because `otherFixedCutClassList` is also specified, the spacings between only the indicated combinations of cut classes must be ≥ 0.02 .

```
spacings(
  ( minViaSpacing "Via1"
    'fixedCutClassList ("c1" "c2" "c3")
    'otherFixedCutClassList ("c4" "c5")
    0.02
  )
);spacings
```



Virtuoso Technology Data Constraints Reference

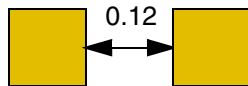
Via Construction Constraints

Example 5: *minViaSpacing* with *exceptExactAligned*

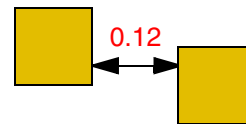
The spacing must be greater than or equal to 0.1 between exactly aligned via cuts and greater than or equal to 0.15 between via cuts that are not exactly aligned.

```
spacings (  
  ( minViaSpacing "Via1"  
    0.1  
  )  
  ( minViaSpacing "Via1"  
    'exceptExactAligned  
    0.15  
  )  
) ;spacings
```

■ Via1



a) PASS. The cuts are exactly aligned and the spacing between them is 0.12 (>0.1).



b) FAIL. The cuts are not exactly aligned, but the spacing between them is 0.12 (<0.15).

Virtuoso Technology Data Constraints Reference

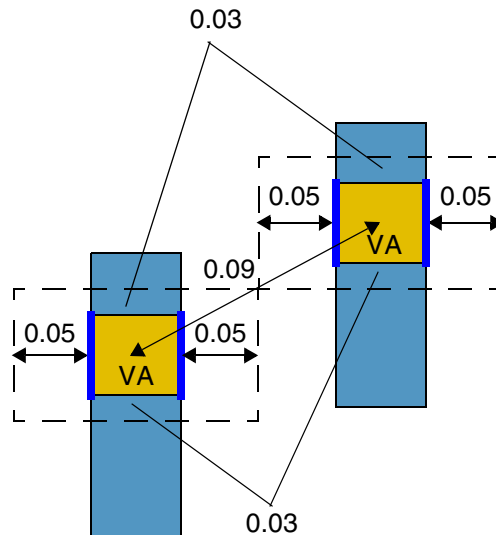
Via Construction Constraints

Example 6: *minViaSpacing* with *centerToCenter*, *cutClass*, *enclosingLayer*, and *sizeByTouchedCorners*

The center-to-center spacing between two VA via cuts must be at least 0.07 if both via cuts have Metal1 extensions less than or equal to 0.02 on a pair of opposite edges and the corners of the two vias touch when the pair of edges that satisfy the specified via edge type is expanded by 0.05 and the other pair of edges is expanded by 0.03.

```
spacings(
  ( viaEdgeType "Via1"
    'anyOppositeExtension 0.02
    1
  )
  ( minViaSpacing "Via1"
    'centerToCenter
    'cutClass "VA"
    'enclosingLayer "Metal1"
    'viaEdgeType 1 'bothCuts
    'sizeByTouchedCorners (0.05 0.03)
    'includeEdgeAligned 'alignedHorizontal
    0.07
  )
);spacings
```

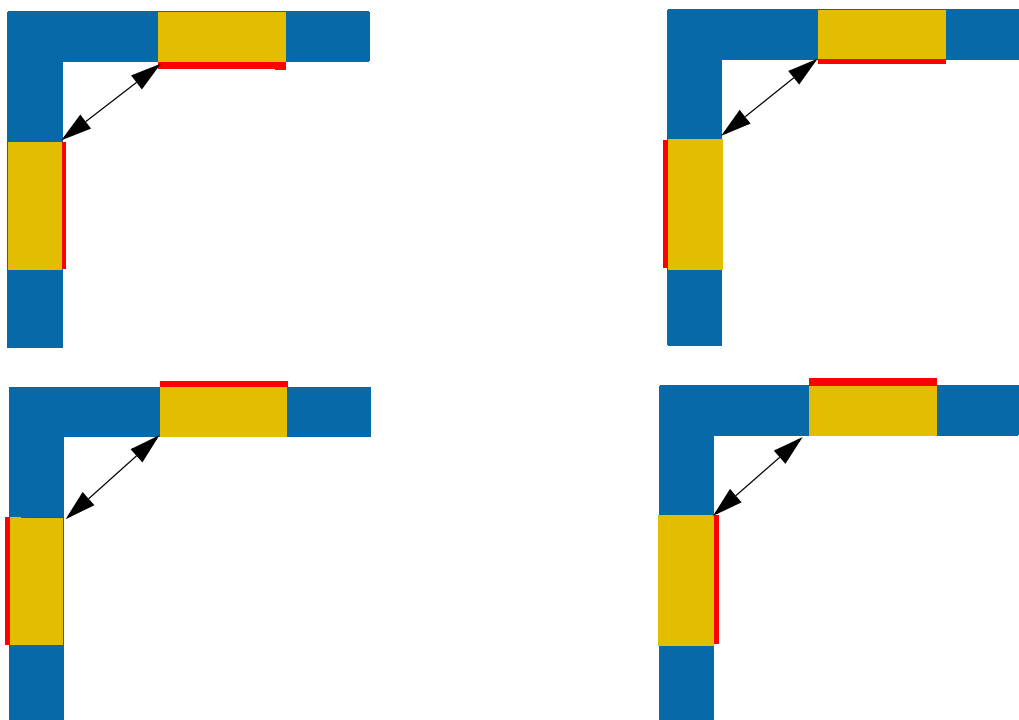
Via1
 Metal1



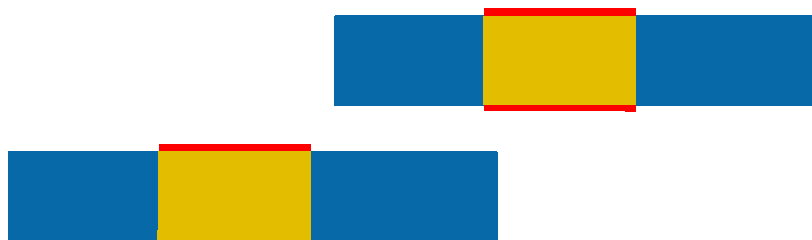
PASS. The constraint applies because both via cuts have a pair of opposite edges with 0.015 Metal1 extension (< 0.02), denoted by the thick blue lines, and when this pair of opposite edges is expanded by 0.05 and the other pair of edges is expanded by 0.03, the corners of the expanded via cuts touch. The constraint is met because the center-to-center spacing between the via cuts is 0.09 (> 0.07).

Example 7: *minViaSpacing* with *orthogonalEdgePair*

The following diagram illustrates cases of orthogonal pairs where the constraint applies. All the distances must be greater than $f\_spacing$. The arrows indicate the distance between the cut shapes having orthogonal pairs of edges.



The following diagram illustrates cases where the constraint will not apply because the edges are parallel and not orthogonal.

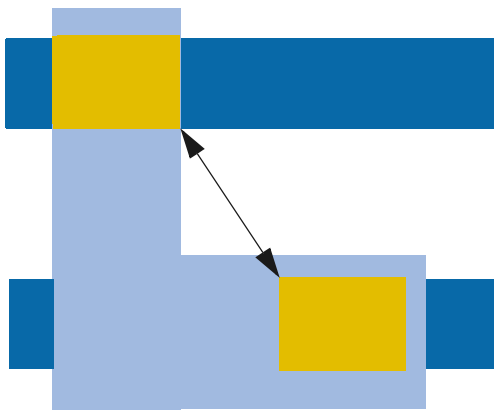


Example 8: *minViaSpacing* with *orthogonalEdgePair* and *orthogonalEnclosingMetal*

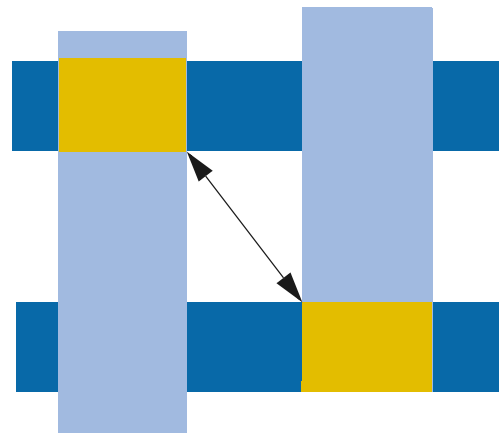
The following example illustrates the use of `orthogonalEdgePair` and `orthogonalEnclosingMetal`.

```
spacings(
  ( minViaSpacing "Via1"
    'enclosingLayer "Metal2"
    'viaEdgeType 1
    'otherViaEdgeType 2
    'orthogonalEdgePair
    'orthogonalEnclosingMetal
    0.5
  )
);spacings
```

Via1
 Metal1
 Metal2



PASS. The constraint applies because the enclosing Metal2 shapes are orthogonal.



The constraint does not apply because the enclosing Metal2 shapes are not orthogonal.

Example 9: *minViaSpacing* with *exceptRanges*

The spacing between a pair of via cuts on Via1 must be at least 0.5. The vias are exempt from the rule if the spacing between them is equal to 0.1 or greater than 0.15 and less than 0.16.

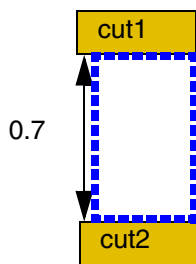
```
spacings(
  ( minViaSpacing "Via1"
    'exceptRanges (0.1 "(0.15 0.16)")
    0.5
  )
);spacings
```

Example 10: minViaSpacing with exceptRanges and facingEdges

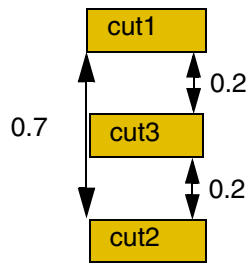
The spacing between a pair of via cuts on Via1 must be at least 0.8. The vias are exempt from the rule if the spacing between them is equal to 0.2 and their edges are not facing each other.

```
spacings (
  ( minViaSpacing "Via1"
    'exceptRanges (0.2)
    'facingEdges
    0.8
  )
) ;spacings
```

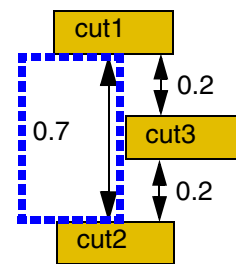
■ Via1



There is a violation between cut1 and cut2 because there is no via between them and spacing is less than 0.8.



There is no violation between cut1 and cut2 because cut3 lies between them. There is no violation between cut2 and cut3 because 0.2 is within the value specified by exceptRanges.



There is a violation between cut1 and cut2 because edges are partially facing each other. There is no violation between cut2 and cut3 because 0.2 is within the value specified by exceptRanges.

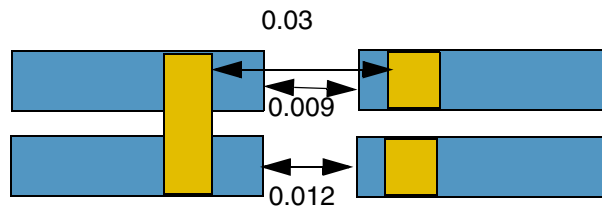
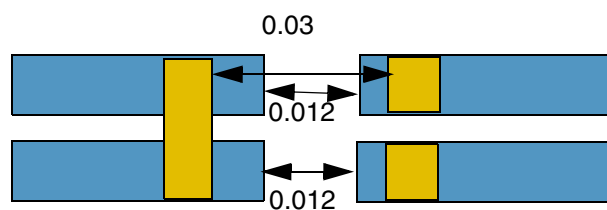
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 11: *minViaSpacing* with *exceptGap* and *exceptSameMetalOverlap*

```
.  
spacings (  
  ( minViaSpacing "V0"  
    'overLayer "M0"  
    'exceptGap 0.012  
    'exceptSameMetalOverlap 0.09  
  )  
) ;spacings
```

■ Via1
■ Metal1



Virtuoso Technology Data Constraints Reference

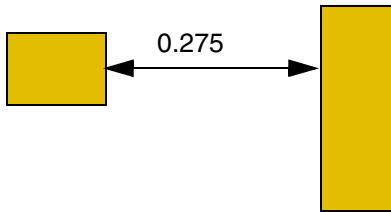
Via Construction Constraints

Example 12: *minViaSpacing* with type of spacing

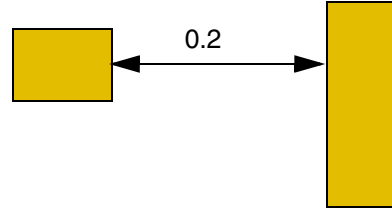
The spacing between two via cuts is checked using the following constraints.

```
spacings(  
  ( minViaSpacing "Via1" 'shortSideToLongSide .275 )  
  ( minViaSpacing "Via1" 'longSideToLongSide 'sameMetal .160 )  
) ;spacings
```

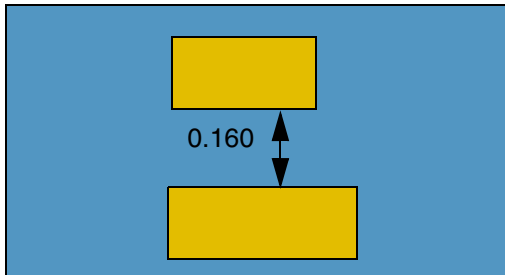
Via1
 Metal1



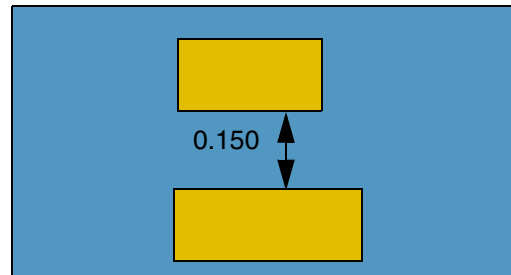
a) PASS. The first constraint applies and passes because the spacing is at least 0.275. The second constraint does not apply.



b) FAIL. The first constraint applies but fails because the spacing is less than 0.275. The second constraint does not apply.



c) PASS. The second constraint applies and passes because the spacing is 0.160. The first constraint does not apply.



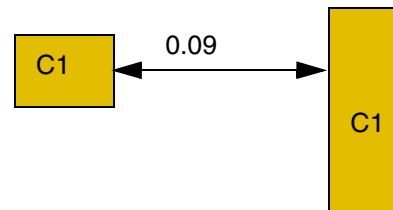
d) FAIL. The second constraint applies and fails because spacing is less than 0.160. The first constraint does not apply.

Example 13: minViaSpacing with overLayer

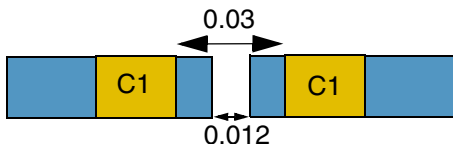
The spacing between two via cuts is checked using the following constraints. Let's assume that all cuts belong to the C1 cut class.

```
("minViaSpacingsCG" nil nil 'and
  spacings(
    ( minViaSpacing "Via1"
      'overLayer "Metal1"
      'exceptGap 0.012
      'exceptSameMetalOverlap
        0.09)
    ) ;spacings
  spacings(
    ( minViaSpacing "Via1"
      'cutClass "C1"
      .03)
    ) ;spacings
  spacings(
    ( minViaSpacing "Via1"
      'cutClass "C2"
      .05)
    ) ;spacings
)
```

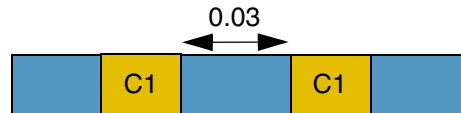
Via1
 Metal1



Constraints 1 and 2 apply to cut class C1 and PASS. Constraint 3 does not apply to cut class C1.



Constraint 1 does not apply because of exceptGap. Constraint 2 applies and is met. Constraint 3 does not apply.



Constraint 1 does not apply because of exceptSameMetalOverlap. Constraint 2 applies and is met. Constraint 3 does not apply.

Virtuoso Technology Data Constraints Reference

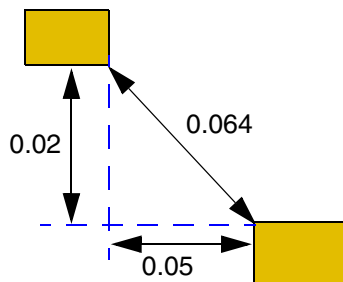
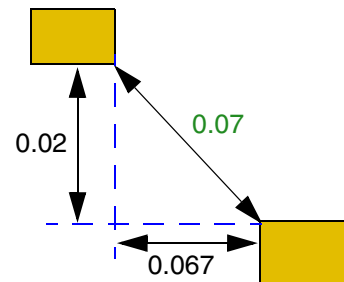
Via Construction Constraints

Example 14: *minViaSpacing* with *verticalSpacingRange* and *horizontalSpacingRange*

Two constraints are specified. The first constraint requires that the vertical spacing between two shapes be less than 0.03 and the horizontal spacing be more than 0.04. The second constraint does not specify vertical or horizontal spacing.

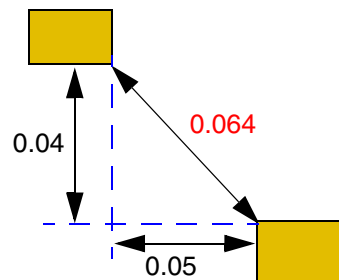
```
spacings (  
  ( minViaSpacing "Via1"  
    'verticalSpacingRange "<0.03"  
    'horizontalSpacingRange ">0.04"  
    0.07  
  )  
  ( minViaSpacing "Via1"  
    0.03  
  )  
) ;spacings
```

■ Via1



b. The first constraint applies because the vertical spacing of 0.02 is < 0.03 and the horizontal spacing of 0.067 is > 0.04 . The constraint passes because the spacing of 0.07 is met. The second constraint passes because the spacing of 0.07 is > 0.03 .

a. The first constraint applies because the vertical spacing of 0.02 is < 0.03 and the horizontal spacing of 0.05 is > 0.04 . The constraint fails because the spacing of 0.064 is < 0.07 . The second constraint passes because the spacing of 0.064 is > 0.03 .



c. The first constraint does not apply because the vertical spacing of 0.04 is not < 0.03 . The second constraint passes because the spacing of 0.064 is > 0.03 .

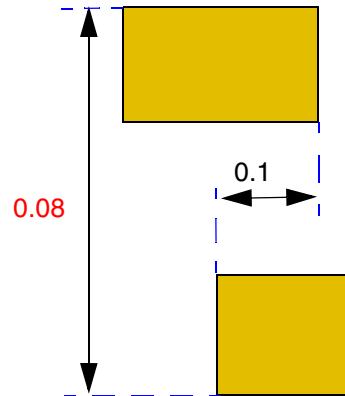
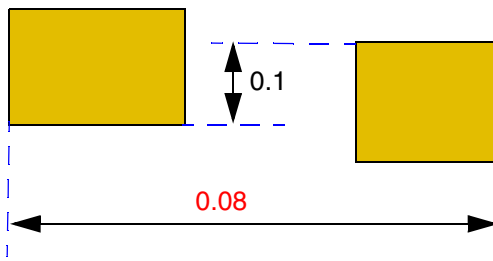
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example 15: minViaSpacing with farEnds

The spacing between the far ends of vias must be greater than or equal to 0.09.

```
spacings(  
  ( minViaSpacing "Via1"  
    'farEnds  
    0.09  
  )  
); spacings
```

■ Via1

FAIL. In both cases, the maximum spacing between two shapes is used as spacing if no direction is specified. The constraint fails because the spacing is less than 0.09.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

minViaSpacing (Two layers)

```
spacings (
  ( minViaSpacing tx_cutLayer1 tx_cutLayer2
    ['centerToCenter]
    ['sameNet | 'sameMetal]
    ['stack]
    ['area f_area]
    ['cutClass { f_width | (f_width f_length) | t_name }]
    ['otherCutClass { f_width | (f_width f_length) | t_name }]|
    ['fixedCutClassList l_fixedCutClassNames
      ['otherFixedCutClassList l_otherFixedCutClassNames]]
  ]
  ['enclosingLayer tx_enclosingLayer
    ['viaEdgeType x_edgeType ['extendBy f_extendBy]
  ]
  ['sizeBy f_sizeBy]
  ['exceptSameNet | 'exceptSameMetal]
  ['overlapNotAllowed]
  ['horizontal | 'vertical
    ['prl f_prl ['oppositeSides] | 'prlRange f_prlRange]
  ['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
    ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    |
    ['outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
  ] |
  ['cornerEuclidian]
  ['deltaVoltage f_voltage]
  f_spacing
)

) ;spacings

spacingTables (
  ( minViaSpacing tx_cutLayer1 tx_cutLayer2
    (( "width" nil nil ["width" nil nil] )
      ['centerToCenter]
      ['sameNet | 'sameMetal]
      ['stack]
      ['area f_area]
      ['cutClass { f_cutWidth | (f_cutWidth f_cutLength) | t_name }]
      ['otherCutClass { f_width | (f_width f_length) | t_name }]
      ['enclosingLayer tx_enclosingLayer
        ['viaEdgeType x_edgeType ['extendBy f_extendBy]
      ]
      ['sizeBy f_sizeBy]
      ['exceptSameNet | 'exceptSameMetal]
      ['overlapNotAllowed]
      ['horizontal | 'vertical
        ['prl f_prl ['oppositeSides] | 'prlRange f_prlRange]
      ]
    )
  )
)
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
[f_default]
['insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
 ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
 |
 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
 ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
 ] |
['cornerEuclidian]
['deltaVoltage f_voltage]
)
(g_table)
)
) ;spacingTables
```

Specifies the minimum spacing required between via cuts on different layers. The required spacing can be independent of the width of the via cuts or can be dependent on the width of one or both via cuts.

Optional parameters determine whether spacing is measured center-to-center or edge-to-edge, whether the constraint applies to a certain connectivity type, whether same-net via cuts on different layers can be stacked if aligned, and whether the constraint applies to via cuts that have an area greater than or equal to the specified value.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_cutLayer1</i>	The first cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer2</i>	The second cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The minimum spacing required between via cuts on different layers, independent of the width of the via cuts.
"width" nil nil ["width" nil nil]	This identifies the index for <i>table</i>.
<i>g_table</i>	The minimum spacing required between via cuts on different layers, depending on the width of the via cuts. The table has the following format: (<i>f_widthCut1</i> <i>f_spaceValue</i>) or (<i>f_widthCut1</i> <i>f_widthCut2</i> <i>f_spaceValue</i>) where, ■ <i>f_widthCut1</i>: The width of the via cut on <i>cutlayer1</i> must be greater than or equal to this value. ■ <i>f_widthCut2</i>: The width of the via cut on <i>cutlayer2</i> must be greater than or equal to this value. ■ <i>f_spaceValue</i>: The spacing between the two cuts on different layers must be greater than or equal to this value. A 1-D table means that the required spacing is dependent on the width of one of the two via cuts, and a 2-D table means that the required spacing is dependent on the width of both via cuts. Type: A 1-D or 2-D table specifying floating-point width and spacing values.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

<code>'centerToCenter</code>	The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge. Type: Boolean
<code>'sameNet 'sameMetal</code>	The connectivity type. If connectivity type is not specified, the constraint applies to any connectivity type, including any two cuts on the same cut layer with no common metal or net. ■ <code>'sameNet</code>: The constraint applies only if the via cuts are on the same net. ■ <code>'sameMetal</code>: The constraint applies only if the via cuts are on contiguous same-metal shapes. Type: Boolean
<code>'stack</code>	The via cuts on two different layers can be stacked if their center points are exactly aligned. Otherwise, the via cuts cannot be stacked.
<code>'area <i>f_area</i></code>	The constraint applies only if the area of a via cut is greater than or equal to this value. The value for this parameter is assumed to be greater than the default spacing value.
<code>'cutClass { <i>f_width</i> (<i>f_width</i> <i>f_length</i>) <i>t_name</i> }</code>	(Virtuoso Advanced Node for Layout Standard) The cut class to which the constraint applies on <i>cutLayer1</i>, specified by width, by width and length, or by name (as defined in a cutClass constraint). ■ <i>f_width</i>: Width ■ <i>f_length</i>: Length ■ <i>t_name</i>: Name of the cut class

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'otherCutClass { f_width | (f_width f_length) | t_name }`

(Virtuoso Advanced Node for Layout Standard) The cut class to which the constraint applies on *cutLayer2*, specified by width, by width and length, or by name (as defined in a *cutClasses* constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

`'fixedCutClassList l_fixedCutClassNames`

The constraint applies to the edges of this list of cut classes if their type is as specified by *x_edgeType*.

`'otherFixedCutClassList l_otherFixedCutClassNames`

The constraint applies to all cut classes in *l_fixedCutClassNames* that have the specified spacing with any cut classes in *l_otherFixedCutClassNames* and are of the type specified by *x_edgeType2*.

If not specified, the constraint applies to all cut classes in *l_fixedCutClassNames*.

`'enclosingLayer tx_enclosingLayer`

(Virtuoso Advanced Node for Layout Standard) The enclosing layer for `'viaEdgeType` extension checks.

Type: String (layer name) or Integer (layer number)

`'viaEdgeType x_edgeType`

(Virtuoso Advanced Node for Layout Standard) The constraint applies only to the via cut edges of this type on *cutLayer1* (the type is defined in the *viaEdgeType* constraint).

`'extendBy f_extendBy`

(Virtuoso Advanced Node for Layout Standard) An extension equal to this value is applied to the via cut edges that satisfy `'viaEdgeType` before the constraint is applied.

If `'sizeBy` is also specified, the extension is applied after sizing the cuts.

`'sizeBy f_sizeBy`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Advanced Node for Layout Standard) The constraint is applied after sizing all the edges of the via cut on *cutLayer1* 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.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'exceptSameNet | 'exceptSameMetal`

(Virtuoso Advanced Node for Layout Only) The constraint does not apply to via cuts with certain connectivity types.

- `'exceptSameNet`: The constraint does not apply to via cuts on the same net.
- `'exceptSameMetal`: The constraint does not apply to via cuts enclosed by a contiguous same-metal shape.

Type: Boolean

`'overlapNotAllowed`

If specified, the constraint applies when vias overlap except when the center points of overlapping vias are aligned with the `stack` parameter being specified.

`'horizontal | 'vertical [ 'prl f_prl ] 'oppositeSides`

(Virtuoso Advanced Node for Layout Only) If a spacing direction is specified for the constraint, then the parallel run length is measured in the orthogonal direction.

When the spacing direction is specified, `f_prl` can also be specified. In this case, the constraint applies only if the parallel run length is equal to or greater than the `f_prl` value, which is measured orthogonally to the spacing direction.

When `f_prl` is specified, the `oppositeSides` parameter can also be specified. In this case, the constraint applies only if the `layer1` shape is surrounded by the `layer2` shape on opposite sides.

`'prlRange f_prlRange`

(Virtuoso Advanced Node for Layout Only) The constraint applies only when the parallel run length between two shapes is in this range. The parallel run length is measured orthogonal to the direction parameter.

`f_default`

The spacing value to be used when no table entry applies.

`'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
| 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The presence or absence of one or more specified layers determines if the constraint applies.

- `'insideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps a shape on one of these layers (*layer3 layer4 ... layerN*).
- `'outsideLayers`: The constraint applies when the region between the shapes on the specified layers (*layer1* and *layer2*) overlaps the region outside the shapes on one of these layers (*layer3 layer4 ... layerN*).

For more information, see [Region-based Rule \(Two layers\)](#).

Type: String (layer name) or Integer (layer number)

`'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'insideLayers` parameter is limited to the specified purposes.

Type: String (purpose name)

`'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)`

The `'outsideLayers` parameter is limited to the specified purposes.

Type: String (purpose name)

`'cornerEuclidian`

(Virtuoso Advanced Node for Layout Only) The corner-to-corner spacing (that is, when parallel run length is less than zero) between shapes is measured using Euclidean measure type (the spacing halo has rounded corners).

The default measure type is Manhattan.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'deltaVoltage f_voltage`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies if both the cut shape and the metal shape have voltages and the delta voltage between the *layer1* shape and the *layer2* shape is greater than this value.

The delta voltage is calculated as follows:

- For asynchronous nets:

$\max(V1_{\max}, V2_{\max}) - \min(V1_{\min}, V2_{\min})$

- For synchronous nets:

$\max(|V1_{\max} - V2_{\max}|, |V1_{\min} - V2_{\min}|)$

Examples

- [Example 1: minViaSpacing with centerToCenter, enclosingLayer, and viaEdgeType](#)
- [Example 2: minViaSpacing with edgeType, enclosingLayer, sizeBy, and extendBy](#)
- [Example 3: minViaSpacing with stack and overlapNotAllowed](#)
- [Example 4: minViaSpacing with direction and prl](#)
- [Example 5: minViaSpacing with direction, prl, and oppositeSides](#)

Virtuoso Technology Data Constraints Reference

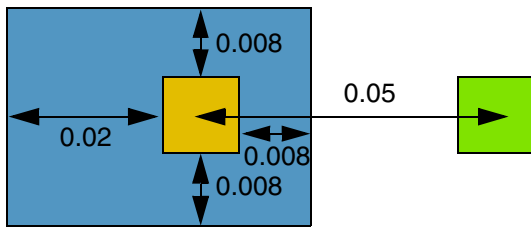
Via Construction Constraints

Example 1: minViaSpacing with centerToCenter, enclosingLayer, and viaEdgeType

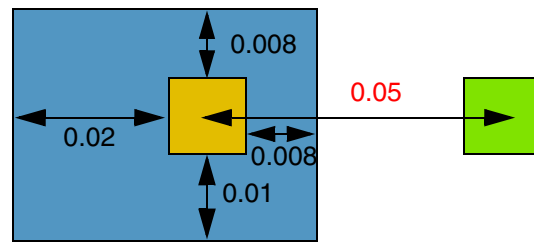
The center-to-center spacing between a Via1 cut that satisfies viaEdgeType 1 (the constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.009) and a Via2 cut must be at least 0.06.

```
spacings (
  ( viaEdgeType "Via1"
    'anyOppositeExtension 0.009
    'negateAnyOppositeExtension
    1
  )
  ( minViaSpacing "Via1" "Via2"
    'centerToCenter
    'enclosingLayer "Metal1"
    'viaEdgeType 1
    0.06
  )
) ;spacings
```

■ Via2
■ Via1
■ Metal1



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).

Virtuoso Technology Data Constraints Reference

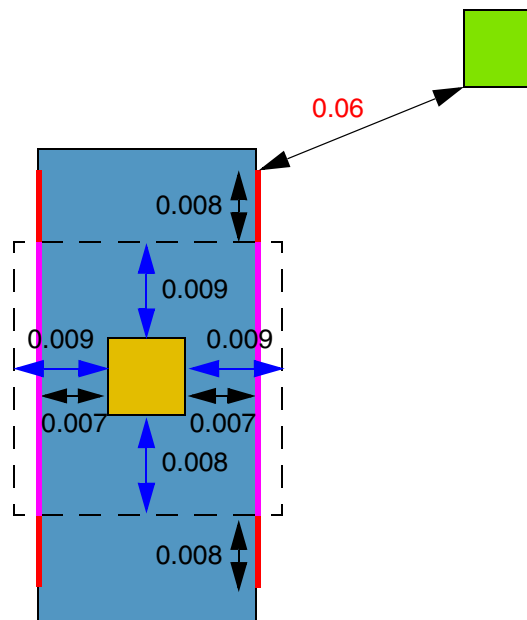
Via Construction Constraints

Example 2: minViaSpacing with edgeType, enclosingLayer, sizeBy, and extendBy

The edge-to-edge spacing between a viaEdgeType 2 Via1 cut edge and a Via2 cut must be at least 0.07 when the Via1 cut is sized by 0.009 and extended by 0.008. The constraint applies only to Via1 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 viaEdgeType 2).

```
spacings(  
  ( viaEdgeType "Via1"  
    'anyOppositeExtension 0.009  
    'edgeExtension 0.009  
    2  
  )  
  ( minViaSpacing "Via1" "Via2"  
    'enclosingLayer "Metal1"  
    'viaEdgeType 2 'extendBy 0.008  
    'sizeBy 0.009  
    0.07  
  )  
) ;spacings
```

■ Via2
■ Via1
■ Metal1



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.

Virtuoso Technology Data Constraints Reference

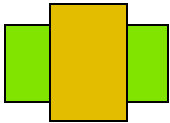
Via Construction Constraints

Example 3: minViaSpacing with stack and overlapNotAllowed

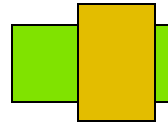
The constraint applies only if the via cuts are overlapping without the center points being aligned.

```
spacings(  
  ( minViaSpacing "Via1" "Via2"  
    'stack  
    'overlapNotAllowed  
    0  
  )  
) ;spacings
```

■ Via2
■ Via1



PASS. The vias are overlapping. However, their centers being aligned, the stack parameter takes precedence.



FAIL. The vias are overlapping. Because the center points of the vias are not aligned, the stack parameter is not applicable.

Virtuoso Technology Data Constraints Reference

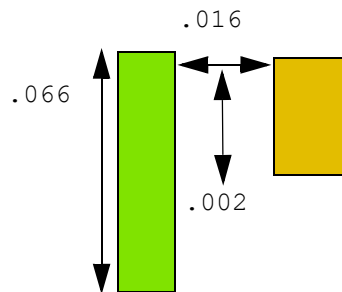
Via Construction Constraints

Example 4: *minViaSpacing* with direction and prl

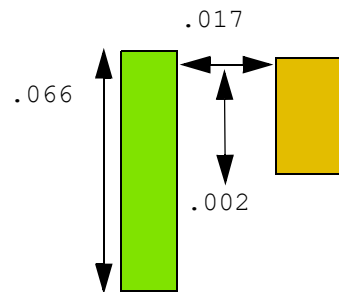
The constraint applies only if the spacing between the vias is more than or equal to .017 and prl is greater than or equal to .002.

```
spacings(  
  ( minViaSpacing "Via1" "Via2"  
    'horizontal  
    'prl .002  
    'cutClass V0_066  
    'exceptSameNet  
    .017  
  )  
); spacings
```

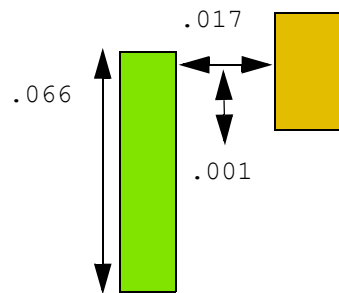
■ Via1
■ Via2



FAIL. The spacing is less than .017.



PASS. The spacing is more than or equal to .017.



The constraint does not apply because prl is less than .002.

Virtuoso Technology Data Constraints Reference

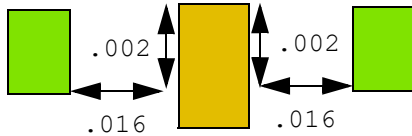
Via Construction Constraints

Example 5: *minViaSpacing* with *direction*, *prl*, and *oppositeSides*

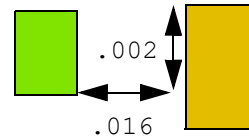
The constraint applies only if the spacing between the vias is more than or equal to .017, prl is greater than or equal to .002, and the Via1 shape is surrounded by the Via2 shape on opposite sides.

```
spacings(  
  ( minViaSpacing "Via1" "Via2"  
    'horizontal  
    'prl .002  
    'oppositeSides  
    'exceptSameNet  
    .017  
  )  
) ;spacings
```

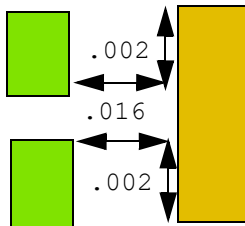
■ Via1
■ Via2



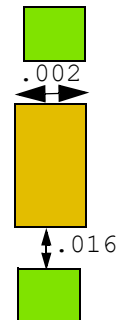
FAIL. Via2 is surrounded on both sides by Via1 but at a spacing of less than .017.



The constraint is not applicable because Via2 is surrounded by only one Via1.



The constraint is not applicable because Via2 is surrounded by Via1 from only one side.



The constraint is not applicable because the spacing is considered only in the horizontal direction.

multiLayer

```
spacings(  
    ( multiLayer tx_Layer  
        'topLayer g_topLayer  
        'bottomLayer g_bottomLayer  
    )  
); spacings
```

(Virtuoso Advanced Node for Layout Only) Specifies a cut layer that crosses multiple routing layers.

Values

<i>tx_layer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Parameters

'topLayer <i>g_topLayer</i>	The top layer that the cut layer connects to.
'bottomLayer <i>g_bottomLayer</i>	The bottom layer that the cut layer connects to.

Example

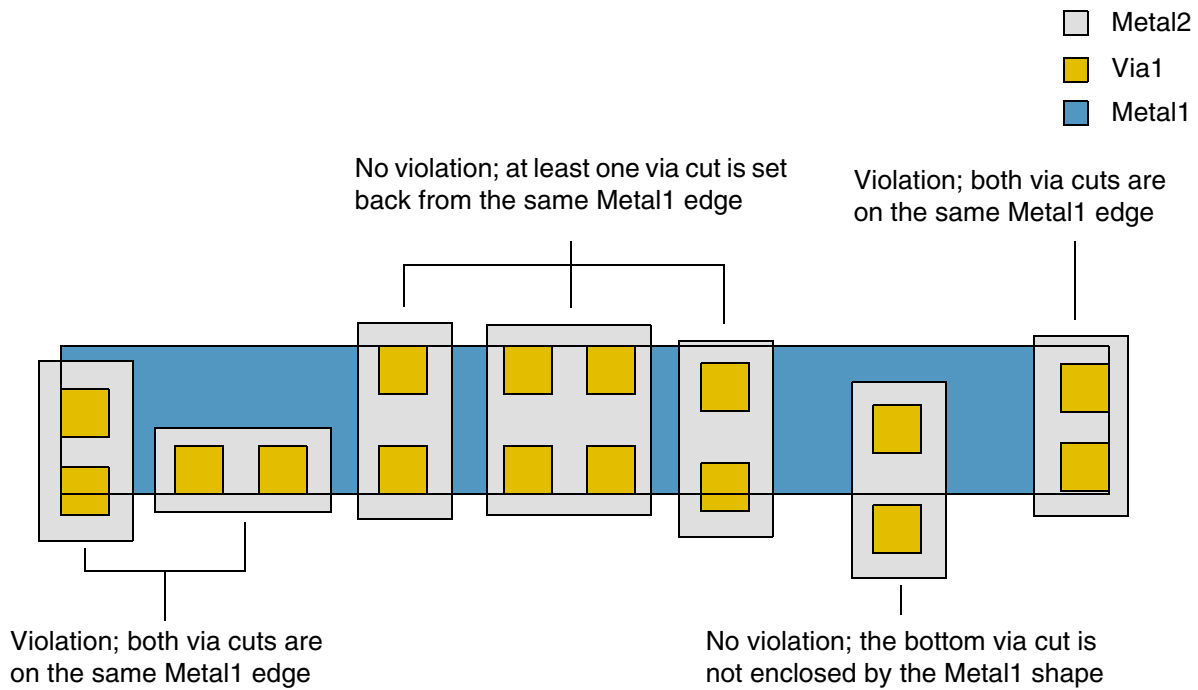
For the cut15 layer, the constraint specifies that it connects to the top layer Metal5 and the bottom layer Metal1.

```
spacings(  
    (multiLayer "cut15"  
        'topLayer "Metal5"  
        'bottomLayer "Metal1"  
    )  
); spacings
```

redundantViaSetback

```
spacingTables (
  ( redundantViaSetback tx_cutLayer tx_metalLayer
    (( "width" nil nil ))
    (g_table)
  )
) ;spacingTables
```

Specifies the minimum *metalLayer* enclosure required when two via cuts are placed on the same edge of the *metalLayer* shape. The required enclosure is dependent on the width of the *metalLayer* shape.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Values

<i>tx_cutLayer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_metalLayer</i>	The metal layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
"width" nil nil	This identifies the index for <i>table</i> .
<i>g_table</i>	The format of a <i>table</i> row is as follows: <i>f_width f_enclosure</i> where, <i>f_width</i> is the width of the metal shape and <i>f_enclosure</i> is the minimum enclosure required when two via cuts are placed on the same edge of the metal shape with width greater than or equal to the corresponding index. Type: A 1-D table specifying floating-point width and enclosure values.

Parameters

None

stackable

```
spacings(  
    ( stackable tx_cutLayer1 tx_cutLayer2  
      {t | nil}  
    )  
); spacings
```

Determines whether the specified cut layers can be stacked.

This constraint applies only to layers with function `cut` or `li`. Layers not identified as stackable by this constraint cannot be stacked.

Values

<i>tx_cutLayer1</i>	The first layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_cutLayer2</i>	The second layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>t nil</i>	If <i>t</i> , the layers can be stacked; if <i>nil</i> , the layers cannot be stacked.

Parameters

None

Example

Layers Via1 and Via2 can be stacked.

```
spacings(  
    ( stackable "Via1" "Via2"  
      t  
    )  
); spacings
```

viaEdgeType

```
spacings (
    ( viaEdgeType tx_layer
        ['edgeExtension f_edgeExt]
        ['anyOppositeExtension f_oppEdgeExt]
        ['negateEdgeExtension]
        ['negateAnyOppositeExtension]
        ['metalWithin (g_metalPr1 g_metalWithin)
            'noMetalWithin (g_noMetalPr1 g_noMetalWithin)
            ['noMetalLowWithin g_noMetalLowWithin]
            {'horizontalEdge | 'verticalEdge}
            ['exceptSpacing g_exceptSpacing
                | 'exceptSpacingRange g_exceptSpacingRange]
            ['sameSide]
        ]
        x_viaEdgeType
    )
) ;spacings
```

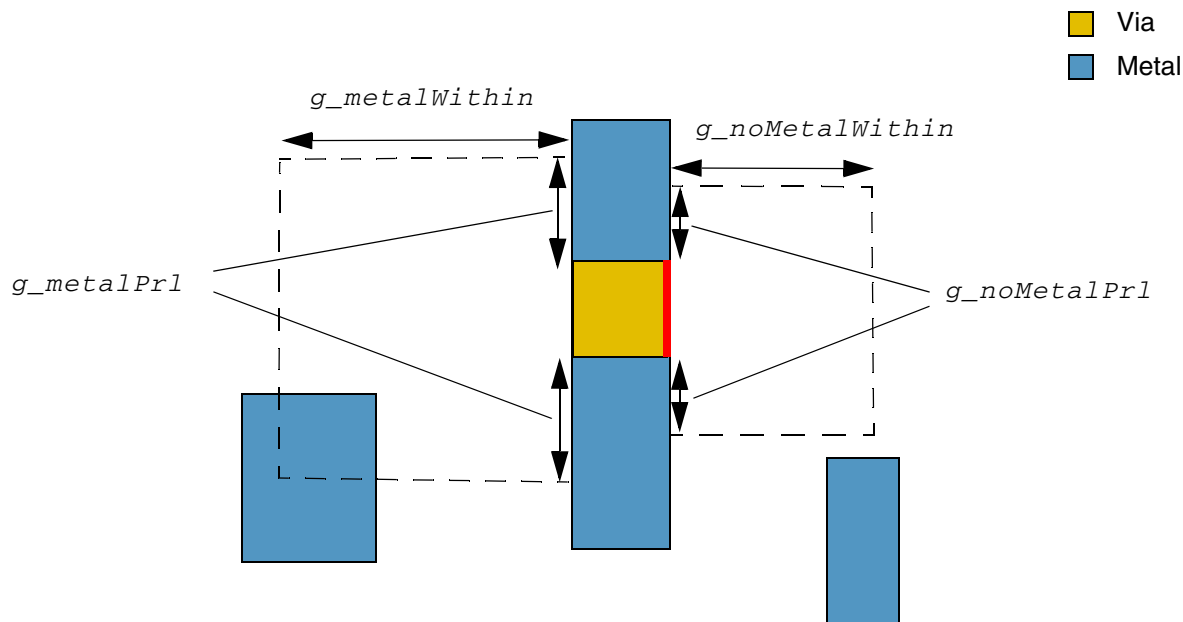
Assigns a type to the edges of a shape on a cut layer. The constraint applies only if the entire length of a cut edge has extension less than or equal to the specified value. The assigned edge type can then be referenced in other constraints.

A cut edge fulfilling `metalWithin` and `noMetalWithin` conditions to any cut should be more than or equal to `g_exceptSpacing`. Search windows are formed on the opposite cut edges in the preferred direction on the above or below metal layers by using `g_metalPr1` and `g_metalWithin` on one of the cut edges and `g_noMetalPr1` and `g_noMetalWithin` on the opposite cut edge. The first metal search window overlaps a metal neighbor whereas the second metal search window does not overlap a metal neighbor. The cut edge without a metal neighbor is subject to this spacing rule.

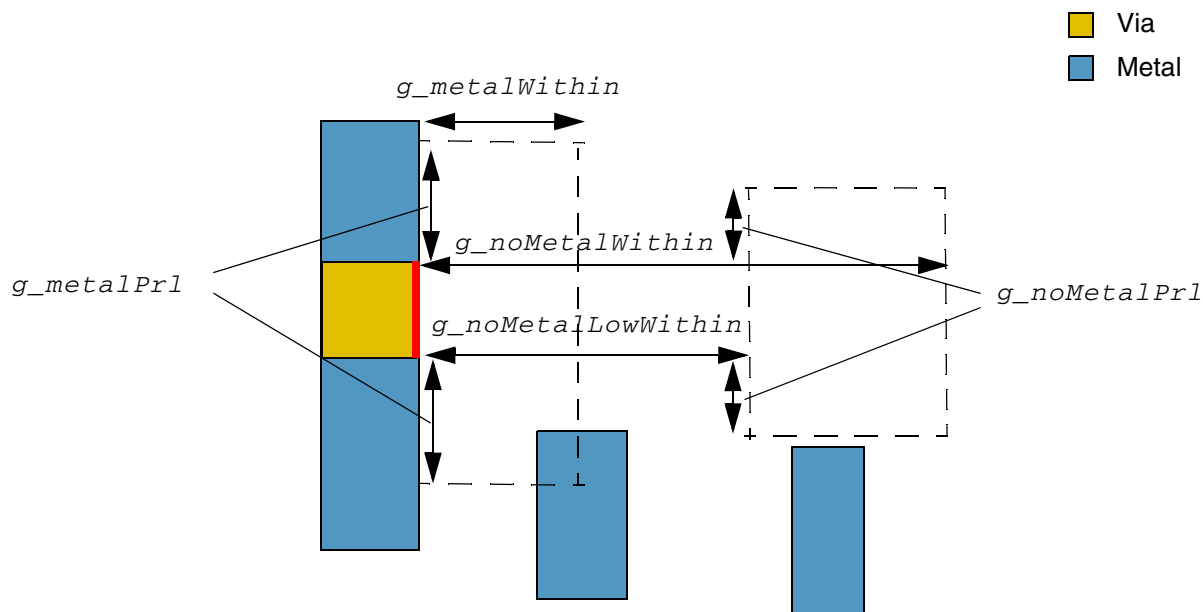
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

In the following illustration, the preferred routing direction of the routing layer is `verticalEdge`. The red cut edge is subject to the cut spacing rules.



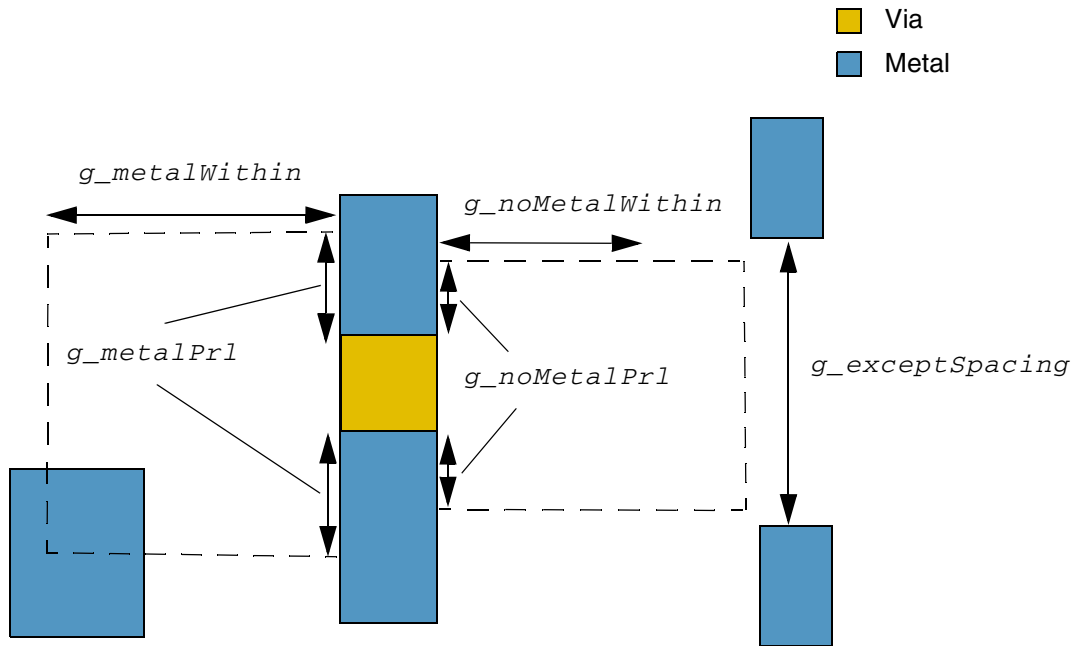
In the following illustration, the preferred routing direction of the routing layer is `verticalEdge`. Both windows are formed on the same cut edge. The red cut edge is subject to the cut spacing rules.



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

In the following illustration, the metal neighbors on the right have a spacing that is equal to *g_exceptSpacing*, which means that spacing rules do not apply to any cut edge in this case.



Values

<i>tx_layer</i>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>x_viaEdgeType</i>	The type assigned to the cut edges.

Parameters

<i>'edgeExtension f_edgeExt</i>	The constraint applies to cut edges with extension less than or equal to this value.
<i>'anyOppositeExtension f_oppEdgeExt</i>	The constraint applies only if a pair of opposite edges have extensions less than or equal to this value.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'negateEdgeExtension`

The constraint applies to cut edges that do not satisfy
`'edgeExtension`.

`'negateAnyOppositeExtension`

The constraint applies only if `'anyOppositeExtension` is not satisfied.

`'metalWithin (g_metalPrl g_metalWithin)`

(Virtuoso Layout Suite EXL and higher tiers) Specifies the metal parallel run length and within values of a cut having a metal neighbor to form a via edge type.

`'noMetalWithin (g_noMetalPrl g_noMetalWithin)`

(Virtuoso Layout Suite EXL and higher tiers) Specifies the metal parallel run length and within values of a cut not having a metal neighbor to form a via edge type.

`'noMetalLowWithin g_noMetalLowWithin`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that no metal must be present in the search window formed between `g_noMetalLowWithin` and `g_noMetalWithin`.

`'horizontalEdge | 'verticalEdge`

(Virtuoso Layout Suite EXL and higher tiers) The cut edge direction to form the metal search windows.

`'exceptSpacing g_exceptSpacing`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that the via edge type would not be formed when the line-end spacing of the neighbor wires is equal to the specified value on the cut edge not having a neighbor overlapping with the search window.

`'exceptSpacingRange g_exceptSpacingRange`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that the via edge type would not be formed when the line-end spacing of the neighbor wires fulfills the specified spacing on the cut edge not having a neighbor overlapping with the search window.

`'sameSide`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that both metal search windows must be formed on the same cut edge.

Examples

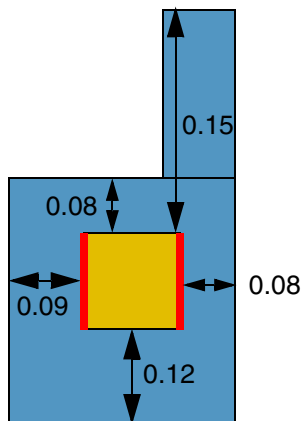
- Example 1: viaEdgeType with edgeExtension
- Example 2: viaEdgeType with edgeExtension and anyOppositeExtension
- Example 3: viaEdgeType with edgeExtension and negateEdgeExtension
- Example 4: viaEdgeType with edgeExtension, anyOppositeExtension, and negateAnyOppositeExtension
- Example 5: viaEdgeType with metalWithin, noMetalWithin, horizontalEdge, and exceptSpacing

Example 1: viaEdgeType with edgeExtension

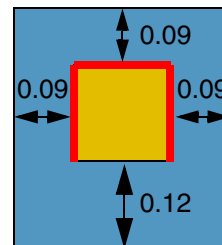
A via cut edge is of type 1 if it has an extension of less than or equal to 0.09 along the entire length of an edge.

```
spacings (
  ( viaEdgeType "Via1"
    'edgeExtension 0.09
    1
  )
) ;spacings
```

■ Via1
■ Metal1



a) The entire length of the left and right edges have extensions less than or equal to 0.09. Therefore, these edges are of type 1. Part of the top edge has a 0.15 extension (>0.09), and therefore, it is not of type 1.



b) The three edges shown in red are of type 1 because these edges have extensions (less than or) equal to 0.09.

Virtuoso Technology Data Constraints Reference

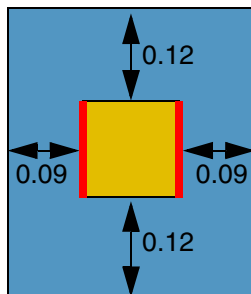
Via Construction Constraints

Example 2: *viaEdgeType* with *edgeExtension* and *anyOppositeExtension*

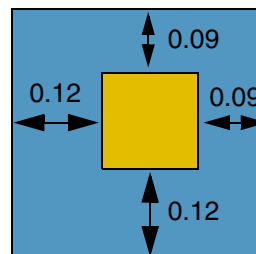
A via cut edge or a pair of opposite via cut edges are of type 2 if they have extensions of less than or equal to 0.09 along the entire length of an edge.

```
spacings(  
  ( viaEdgeType "Via1"  
    'edgeExtension 0.09  
    'anyOppositeExtension 0.09  
    2  
  )  
) ;spacings
```

■ Via1
■ Metal1



a) The left and right edges have extensions (less than or) equal to 0.09. Therefore, these edges are of type 2.



b) The constraint does not apply because the via cut does not have two opposite edges with extension less than or equal to 0.09.

Virtuoso Technology Data Constraints Reference

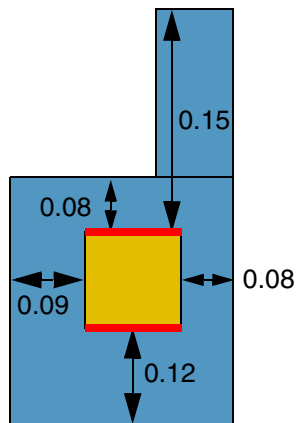
Via Construction Constraints

Example 3: *viaEdgeType* with *edgeExtension* and *negateEdgeExtension*

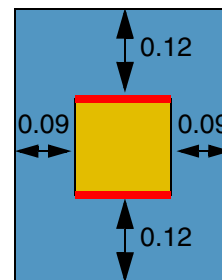
A via cut edge is of type 3 if it has an extension of greater than 0.09 along the entire length of an edge.

```
spacings(  
  ( viaEdgeType "Via1"  
    'edgeExtension 0.09  
    'negateEdgeExtension  
    3  
  )  
); spacings
```

■ Via1
■ Metal1



a) Part of the top edge has extension 0.15 and the bottom edge has extension 0.12 (both >0.09). Therefore, these edges are of type 3.



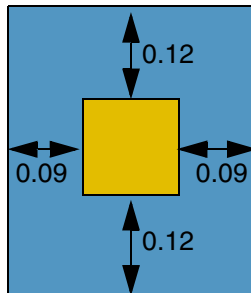
b) The top and bottom edges have extension 0.12 (>0.09), and are, therefore, of type 3.

Example 4: *viaEdgeType* with *edgeExtension*, *anyOppositeExtension*, and *negateAnyOppositeExtension*

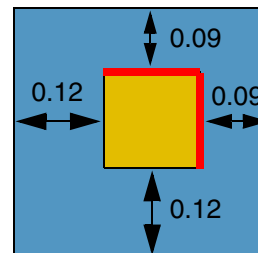
A via cut edge is of type 4 if it has an extension of less than or equal to 0.09 along the entire length of the edge. A pair of opposite via cut edges are of type 4, if they have extensions greater than 0.09 along the entire length of the edge.

```
spacings (
  ( viaEdgeType "Via1"
    'edgeExtension 0.09
    'anyOppositeExtension 0.09
    'negateAnyOppositeExtension
      4
  )
) ;spacings
```

■ Via1
■ Metal1



a) The constraint does not apply because the via cut has two opposite edges with extensions (less than or) equal to 0.09.



b) The edges in red are adjacent edges and have extension (less than or) equal to 0.09. Therefore, these edges are of type 4.

Example 5: *viaEdgeType* with *metalWithin*, *noMetalWithin*, *horizontalEdge*, and *exceptSpacing*

A via cut edge is of type 1 if the *metalWithin* and *noMetalWithin* values are as specified and the cut edge direction is horizontal, except when the line-end spacing of the neighbor wires is 0.1 on the cut edge not having a neighbor overlapping with the search window.

```
spacings (
  (viaEdgeType "Via2"
    'metalWithin (0.03 0.04) 'noMetalWithin (0.06 0.08)
    'horizontalEdge
    'exceptSpacing 0.1
    1
  )
) ;spacings
```

viaGroup

```

spacings (
    ( viaGroup tx_layer
        'groupName t_groupName
        ['cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
        | 'cutClassGroupList f_cutClassNames]
        ['prlRangeOne g_prlRangeOne 'prlRangeTwo g_prlRangeTwo]
        'dualDistance (g_minDistance g_maxDistance)
        ['mixedCutClasses]
        ['sameMask]
        ['allowBend
            ['disallowHorizontal | 'disallowVertical]
        ['spacingPrl (g_prl g_spacing)
            ['spacingPrlHorizontal | 'spacingPrlVertical]
        ]
        g_cuts
    )
) ;spacings

```

(Virtuoso Advanced Node for Layout Only) Specifies the maximum number of cuts in a via group when the parallel run length is within the `prlRangeOne` range on one side and the `prlRangeTwo` range on the other side and the spacing is in the `dualDistance` range. In addition, all cuts must be diagonal, without any bend or notch.

Values

<code>tx_layer</code>	The cut layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<code>g_cuts</code>	The maximum number of cuts.

Parameters

```

'groupName t_groupName
    Name of the via group.

'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name} |
'cutClassGroupList f_cutClassNames

```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The constraint applies to the cut class specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class
- *f_cutClassNames*: (Virtuoso Layout Suite EXL and higher tiers) Any cuts belonging to the list of cut classes form a via group. Each entry is specified with two values that specify the dimensions of a cut.

`'prlRangeOne g_prlRangeOne 'prlRangeTwo g_prlRangeTwo`

Any two cuts in the via group with parallel run length in the range `prlRangeOne` in one direction and in the range `prlRangeTwo` in the other direction.

Type: Floating-point values specifying a [range](#) of parallel run lengths

`'dualDistance (g_minDistance g_maxDistance)`

Any two cuts in the via group must have a spacing of more than *g_minDistance* and less than *g_maxDistance*.

`'mixedCutClasses` (Virtuoso Layout Suite EXL and higher tiers) The constraint specifies that mixed cuts with different cut classes can form a via group.

`'sameMask` The constraint applies only to via cuts on the same mask.

Type: Boolean

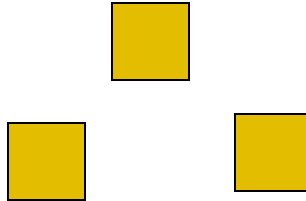
`'allowBend ['disallowHorizontal | 'disallowVertical]`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Layout Suite EXL and higher tiers) When *g_cuts* is 3 and *allowBend* is specified, the constraint applies to a group of three via cuts that form a bend or notch pattern.

For example, unless *allowBend* is specified, the following set of vias cannot form a via group because they form a bend.



The optional directional parameters *disallowHorizontal* and *disallowVertical* indicate that bends in a specified direction with two cuts having PRL more than 0 in the same direction are not allowed.

```
'spacingPrl (g_prl g_spacing)
```

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies when the cuts have spacing greater than *g_spacing* and the parallel run length less than *g_prl*.

```
'spacingPrlHorizontal | 'spacingPrlVertical
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies the direction of the spacing measurement in *spacingPrl*.

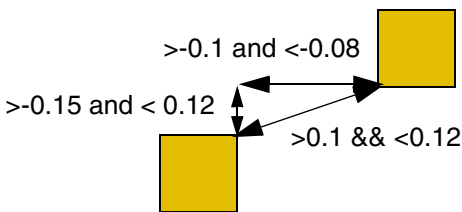
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

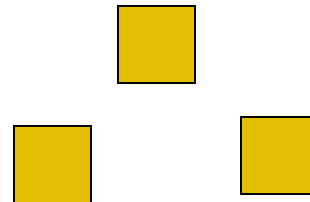
Example

The maximum number of cuts for cutClass VA in viaGroup VGA is 3. The cuts in the group need to meet prlRangeOne, prlRangeTwo, and dualDistance.

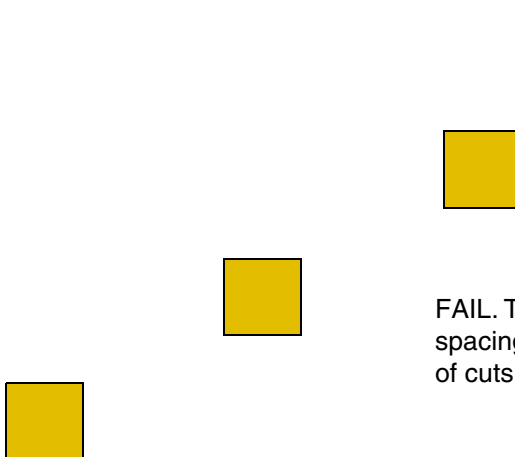
```
spacings(  
  (viaGroup "Via1"  
    'groupName "VGA"  
    'cutClass "VA"  
    'prlRangeOne "(-0.1 -0.08)"  
    'prlRangeTwo "(-0.15 0.12)"  
    'dualDistance (0.1 0.12)  
    3)  
  )  
) ;spacings
```



a) PASS. The cuts are exactly aligned and the PRL and spacing conditions are met.



FAIL. Assuming that PRL and spacing conditions are met, this set of vias cannot form a group because they form a notch or bend.



FAIL. The cuts are exactly aligned and the PRL and spacing conditions are met. However, the number of cuts exceeds 3.

viaKeepoutMetal

```
orderedSpacings (
    ( viaKeepoutMetal tx_cutLayer tx_metalLayer
      'horizontal | 'vertical
      ['subtractTrim]
      ['cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}]
      f_extension
    )
) ;orderedSpacings
```

(Virtuoso Layout Suite EXL and higher tiers) Specifies a keepout region extending in a specified direction from the edges of a via layer. Any metal not touching the cut shape and overlapping or touching the keepout region would be a violation of the constraint.

Values

<i>tx_cutLayer</i>	The via layer to which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>tx_metalLayer</i>	The metal layer to which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_extension</i>	Cut edges in the specified direction are extended outward by this value in Euclidean measurement to define a keepout region.

Parameters

'horizontal 'vertical	The direction in which the extension is measured. ■ 'horizontal: The edges are extended horizontally. ■ 'vertical: The edges are extended vertically. Type: Boolean
'subtractTrim	The trim is subtracted from the layer, if it is trimmable, before checking. Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

```
'cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}
```

The constraint applies only to the cut classes specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name

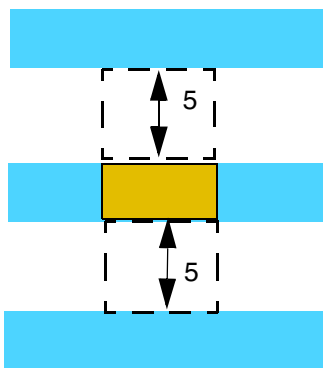
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

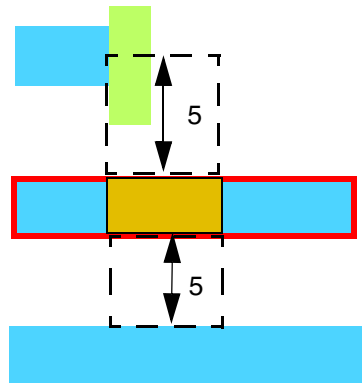
Example

```
orderedSpacings (  
  (viaKeepoutMetal "Via1" "M0"  
    'vertical  
    'cutClass "viaBar"  
    'subtractTrim  
    5)  
)  
);orderedSpacings
```

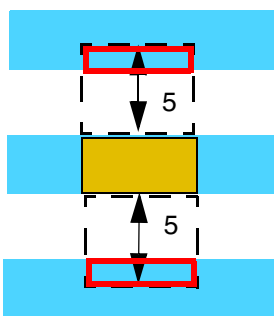
■ Via1
■ M0
■ Trim Shape



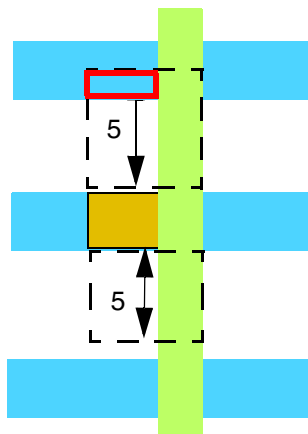
a) PASS. There are no metal shapes in the keepout region.



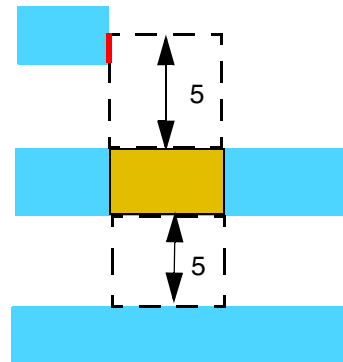
b) PASS. The other metal in the region is covered by the trim shape.



c) FAIL. The metal shapes overlap the keepout region.



d) FAIL. The trim shape does not cover a portion of the other metal.



e) FAIL. The metal touches the edge of the keepout region.

viaMeshEscapeLayer

```
spacings (
    ( viaMeshEscapeLayer tx_layer )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the layer up to which a stranded via has to be constructed.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)

Example

The stranded via has to be constructed up to layer Metal3.

```
spacings (
    ( viaMeshEscapeLayer "Metal3" )
) ;spacings
```

viaMeshMinCuts

```
spacings(  
    ( viaMeshMinCuts tx_layer  
      ['viaMeshStrandLayer tx_layer 'viaMeshStrandNumber x_strandNum]  
      x_minCuts )  
    ) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies the number of cuts needed to effectively connect to a multi-finger pin using a mesh structure.

Values

<i>x_minCuts</i>	The minimum number of cuts needed.
	Type: Integer

Parameters

<i>tx_layer</i>	The layer on which the constraint is applied.
'viaMeshStrandLayer <i>tx_layer</i>	
	The layer on which the number of strands is specified.
'viaMeshStrandNumber <i>x_strandNum</i>	
	The number of strands needed on the specified layer.

Example

Layer `Via1` requires a minimum of 2 cuts and must connect to 2 strands of `Metal2`.

```
spacings(  
    ( viaMeshMinCuts "Via1"  
      'viaMeshStrandLayer "Metal2"  
      'viaMeshStrandNumber 2  
      2  
    )  
    ) ;spacings
```

viaRequiredWithin

```
spacings (
  ( viaRequiredWithin tx_layer
    ['cutClass { f_width | (f_width f_length) | t_name }']
    f_spacing
  )
) ;spacings
```

(Virtuoso Advanced Node for Layout Standard) Specifies that a via cut must have a neighboring via cut within the given distance.

Values

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_spacing</i>	The spacing between two neighboring via cuts must be less than this value.

Parameters

<code>'cutClass { f_width (f_width f_length) t_name }</code>	The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a cutClasses constraint).
■ <i>f_width</i> : Width	
■ <i>f_length</i> : Length	
■ <i>t_name</i> : Name of the cut class	

Virtuoso Technology Data Constraints Reference

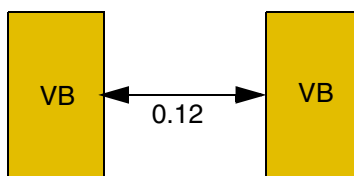
Via Construction Constraints

Example

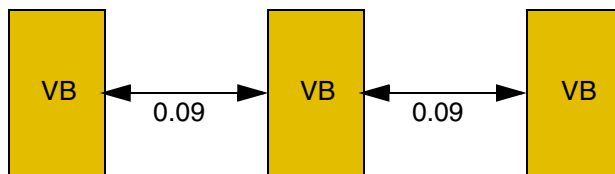
The distance between two neighboring VB via cuts on Via1 must be less than 0.1.

```
spacings(  
  ( viaRequiredWithin "Via1"  
    'cutClass "VB"  
    0.1  
  )  
) ;spacings
```

 Via1



a) FAIL. The spacing between the two via cuts is 0.12 (>0.1).



b) PASS. The via cuts are spaced at 0.09 (<0.1).

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

viaSpacing

```
spacings (
  ( viaSpacing tx_cutLayer
    ['centerToCenter]
    ['sameNet | 'sameMetal | 'sameMetalOverlap | 'sameVia | 'sameMetalLayer
    g_sameMetalLayer]
    ['exceptSamePGNet]
    ['cutClass {f_width | (f_width f_length) | t_name}
      ['sideParaOverlap] ['allCuts]
    ]
    [['exactAligned x_numAlignedCuts ['exactSpacing]]
    | ['noPrl ['horizontalEdge | 'verticalEdge]]
    | ['prl ['prlHorizontal | 'prlVertical] ['sameSide]]
    ]
    ['cutSizeRanges (f_xRange f_yRange)
      | 'cutSizeRanges (f_xRange1 f_yRange1 f_xRange2 f_yRange2)
    ]
    ['exceptOppositeCornerNeighbors]
    ['twoCuts x_twoCuts
      ['twoCutsSpacing g_twoCutsSpacing]
      ['sameNeighborCuts]]
    ['sameMask]
    ['minDistance f_lowDistance]
    ['exceptWithin f_exceptWithin]
    ['edgeExtension f_edgeExtension 'layer g_layer
      {'horizontal | 'vertical}
      ['metalMask1 | 'metalMask2 | 'metalMask3 | 'metalMask4]
      ['width g_width]
      ['exceptLineEndGap g_exceptLineEndGap]
      ['anyCut] ['exceptSameMetalOverlap]
    ]
    'numCuts x_numCuts
    'distance f_distance
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
      | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      ['outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)]
    ]
    f_spacing
  )
) ;spacings
```

Specifies the required minimum spacing between adjacent via cuts. Many processes require a larger spacing between via cuts when a via cut has three or more adjacent via cuts. This ensures that geometries are not merged during fabrication.

A pair of via cuts is considered adjacent if the distance between the via cuts in any direction is less than the specified *distance* value. This includes the distance measured along a 45-

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

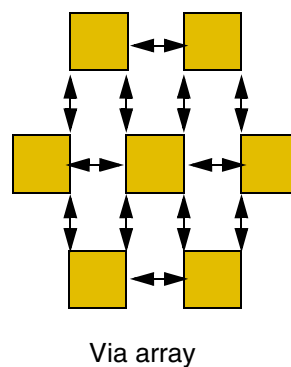
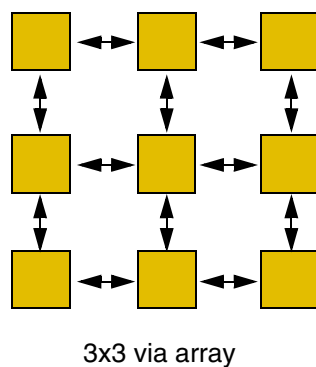
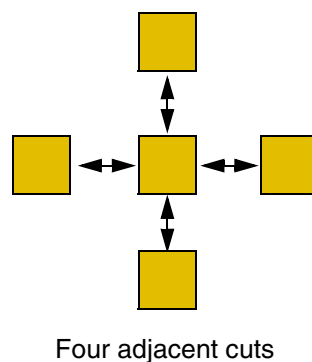
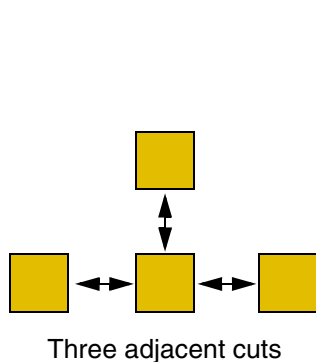
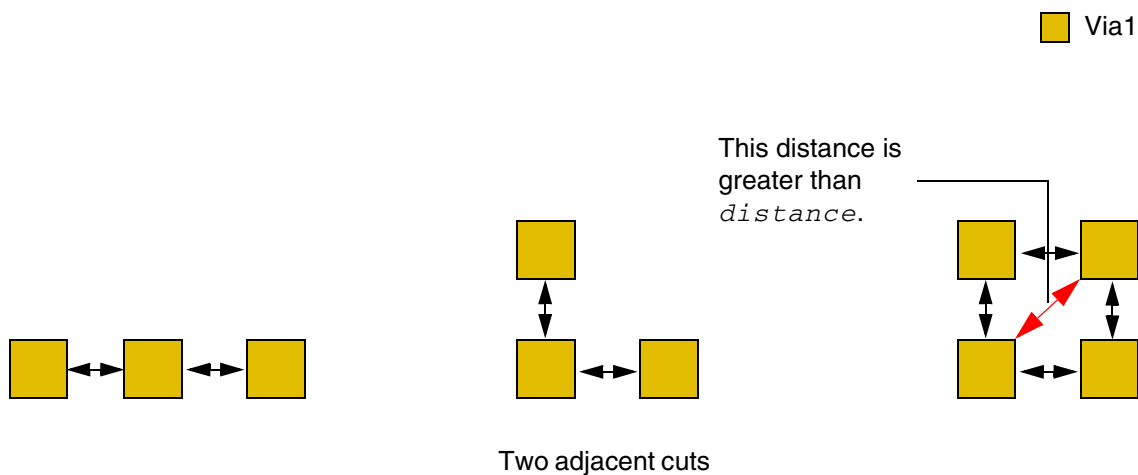
degree angle. Additionally, a via cut must have a certain minimum number of adjacent via cuts for the constraint to apply.

Optional parameters determine whether the spacing is measured center-to-center or edge-to-edge, whether the constraint applies to a certain connectivity type, and whether the constraint applies to via cuts on power and ground nets. In some processes, the constraint applies only if the via cuts have a parallel run length less than zero, and, in other processes, the constraint applies only if the via cuts are exactly aligned. You can also use this constraint to define the spacing for a cut class.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

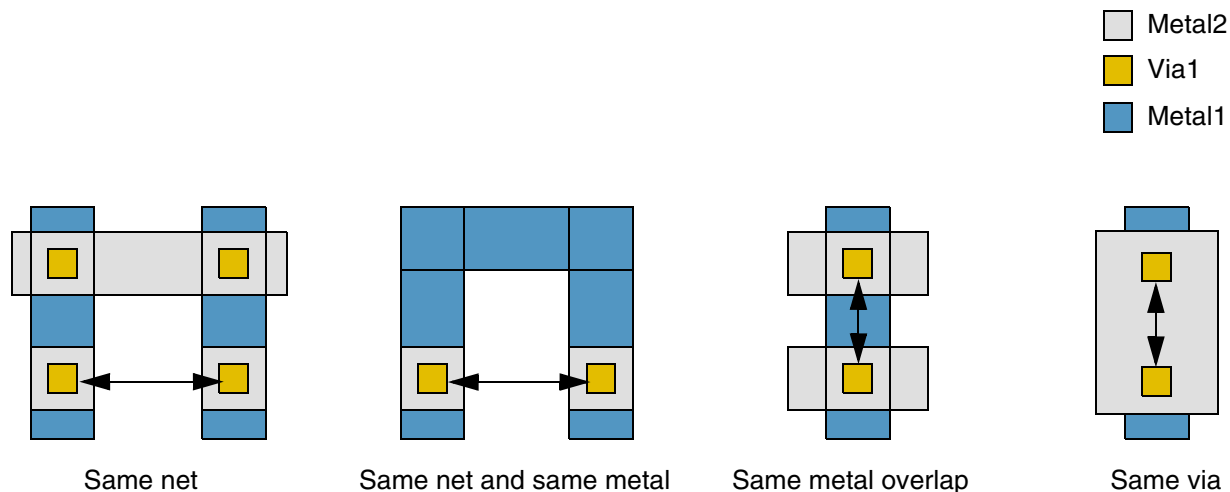
The following figures illustrate the concept of adjacency. The distance between two adjacent via cuts must be less than or equal to *distance*, denoted by the black arrows in the figures below:



Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The following figures illustrate the different connectivity types:



Values

tx_cutLayer

The layer on which the constraint is applied.

Type: String (layer and purpose names) or Integer (layer number)

f_spacing

The minimum spacing requirement that must be satisfied.

Parameters

'centerToCenter

The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'sameNet | 'sameMetal | 'sameMetalOverlap | 'sameVia |
'sameMetalLayer *g_sameMetalLayer*

The connectivity type. If connectivity type is not specified, the constraint applies to any connectivity type, including any two cuts on the same cut layer with no common metal or net.

- 'sameNet: The constraint applies only if the via cuts are on the same net.
- 'sameMetal: The constraint applies only if the via cuts are on contiguous same-metal shapes.
- 'sameMetalOverlap: The constraint applies only if the via cuts are overlapped by a single metal shape.
- 'sameVia: The constraint applies only if the via cuts are overlapped by a single metal shape from above and by another single metal shape from below.
- 'sameMetalLayer: The constraint applies only if the via cuts are on contiguous same metal shapes on the layer specified by this parameter. This parameter is dependent on sameMetal and sameMetalOverlap.

Type: Boolean

'exceptSamePGNet

The constraint does not apply if the via cuts are on the same power or ground net.

Type: Boolean

'cutClass { *f_width* | (*f_width* *f_length*) | *t_name* }

The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a [cutClasses](#) constraint).

- *f_width*: Width
- *f_length*: Length
- *t_name*: Name of the cut class

'sideParaOverlap

The constraint applies only if the long edges of rectangular via cuts have parallel run length less than zero.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

'allCuts

(Virtuoso Advanced Node for Layout Standard) The neighboring via cuts can belong to any cut class.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'exactAligned x_numAlignedCuts`

The constraint applies if a via cut has at least this many exactly aligned, horizontally and/or vertically, adjacent via cuts, or if the number of adjacent via cuts is greater than or equal to the number of via cuts specified with `'numCuts`, of which at least one is not exactly aligned.

The number of via cuts specified with this parameter must be greater than the number of via cuts specified with the `'numCuts` parameter.

`'exactSpacing`

The spacing between the exactly aligned via cuts must be equal to the constraint value.

Type: Boolean

`'noPr1`

The constraint applies only between via cuts with parallel run length less than zero.

Type: Boolean

`'horizontalEdge | 'verticalEdge`

(Virtuoso Layout Suite EXL and higher tiers) The direction of the cut with parallel run.

`'pr1`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that the parallel run length (PRL) of two cuts is greater than the specified value.

`'pr1Horizontal | 'pr1Vertical`

(Virtuoso Layout Suite EXL and higher tiers) Specifies the direction of the cut PRL.

`'sameSide`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that the neighbor cuts should be on the same side of the checking cut fulfilling extension specified in the `edgeExtension` parameter that should also be specified.

`'cutSizeRanges (f_xRange f_yRange)`

`| 'cutSizeRanges (f_xRange1 f_yRange1 f_xRange2 f_yRange2)`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

(Virtuoso Advanced Node for Layout Standard) When two ranges, (*xRange yRange*), are specified, the constraint applies only if all the via cuts have *xSpan* in range *xRange* and *ySpan* in *yRange*.

When four ranges, (*xRange1 yRange1 xRange2 yRange2*), are specified, the constraint applies only if the first via cut has *xSpan* in range *xRange1* and *ySpan* in *yRange1*, and the adjacent via cuts have *xSpan* in range *xRange2* and *ySpan* in *yRange2*.

'exceptOppositeCornerNeighbors

(Virtuoso Advanced Node for Layout Standard) The constraint does not apply if the adjacent via cuts are on opposite corners.

Type: Boolean

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'twoCuts x_twoCuts`

(Virtuoso Advanced Node for Layout Standard) The constraint applies between two via cuts if each of these via cuts has this many adjacent via cuts, or if the number of adjacent via cuts is greater than or equal to the number of via cuts specified with `'numCuts`.

The number of via cuts specified with this parameter must be less than the number of via cuts specified with the `'numCuts` parameter.

`'twoCutsSpacing g_twoCutsSpacing`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the cuts specified with `twoCuts` have a spacing less than this value.

`'sameNeighborCuts`

(Virtuoso Advanced Node for Layout Only) The via cuts that meet the `'twoCuts` requirement must have the same adjacent via cuts as neighbors.

Type: Boolean

`'sameMask`

(Virtuoso Advanced Node for Layout Only) The constraint applies only to via cuts on the same mask.

Type: Boolean

`'minDistance f_lowDistance`

(Virtuoso Advanced Node for Layout Only) The distance between two via cuts must be greater than or equal to this value and less than `distance` for the via cuts to qualify as adjacent via cuts.

`'exceptWithin f_exceptWithin`

(Virtuoso Advanced Node for Layout Only) The constraint does not apply if all adjacent via cuts are at a distance less than this value.

`'edgeExtension f_edgeExtension`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the via cut has an extension less than the given value. The extension must be in the direction specified by `'vertical` | `'horizontal` and on the layer specified by `'layer`.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

`'layer g_layer`

(Virtuoso Advanced Node for Layout Only) The constraint applies only for cuts on the specified layer. See also `'edgeExtension`.

`'horizontal | 'vertical`

(Virtuoso Advanced Node for Layout Only) The constraint applies only if the extension is in the specified direction. See also `'edgeExtension`.

`'metalMask1 | 'metalMask2 | 'metalMask3 | 'metalMask4`

(Virtuoso Layout Suite EXL and higher tiers) This parameter specifies on which metal mask, `metalMask1`, `metalMask2`, `metalMask3`, or `metalMask4`, the metal containing the cut is applied.

`'width g_width`

(Virtuoso Layout Suite EXL and higher tiers) The constraint applies only if a wire containing a cut that the extension condition is applied on has width less than or equal to `g_width`.

`'exceptLineEndGap g_exceptLineEndGap`

(Virtuoso Layout Suite EXL and higher tiers) The `f_edgeExtension` value does not apply if the `lineEndGap` constraint condition on the adjacent metal layer is met.

`'anyCut`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that the extension condition is triggered on any triggering or neighbor cut.

`'exceptSameMetalOverlap`

(Virtuoso Layout Suite EXL and higher tiers) Specifies that neighbor cuts sharing a common metal are counted as individual cuts.

`'numCuts x_numCuts`

The constraint applies only if the number of adjacent via cuts is greater than or equal to this value.

If the number of via cuts is less than this value, the `minSpacing` constraint applies.

`'distance f_distance`

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

The constraint applies only if the distance between the via cuts is less than this value. All such via cuts are considered adjacent via cuts.

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
```

Determines if the constraint applies, based on the presence or absence of one or more layers.

- 'insideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap a shape on one of these layers (*layer1 layer2 ... layerN*).
- 'outsideLayers: The constraint applies only if the shapes on the specified layer (*layer*) overlap the region outside the shapes on one of these layers (*layer1 layer2 ... layerN*).

For more information, see [Region-based Rule \(One layer\)](#).

Type: String (layer name) or Integer (layer number)

```
'insidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'insideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

```
'outsidePurposes (t_purpose1 t_purpose2 ... t_purposeN)
```

The 'outsideLayers parameter is limited to only the specified purposes.

Type: String (purpose name)

Old Syntax

```
spacings( ( viaSpacing tx_cutLayer g_spacingDefinition ) ... )
```

where,

g_spacingDefinition is specified as *f_spacing* or (*x_numCuts f_distance f_spacing*)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example

```
spacings (
  ( viaSpacing "Via1" 0.5 )
  ( viaSpacing "Cont1" (3 1.5 1.0) )
) ;spacings
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Examples

- Example 1: viaSpacing with centerToCenter and noPrI
- Example 2: viaSpacing with centerToCenter and exactAligned
- Example 3: viaSpacing with centerToCenter and twoCuts
- Example 4: viaSpacing with centerToCenter, twoCuts, and sameNeighborCuts
- Example 5: viaSpacing with cutClass and noPrI
- Example 6: viaSpacing with cutClass, allCuts, and noPrI
- Example 7: viaSpacing with cutClass and sideParaOverlap
- Example 8: viaSpacing with cutSizeRanges and noPrI
- Example 9: viaSpacing with cutSizeRanges and exceptOppositeCornerNeighbors
- Example 10: viaSpacing with exceptWithin, numCuts, and distance
- Example 11: viaSpacing with edgeExtension
- Example 12: viaSpacing with twoCutsSpacing
- Example 13: viaSpacing with width

Virtuoso Technology Data Constraints Reference

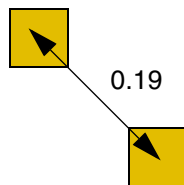
Via Construction Constraints

Example 1: viaSpacing with centerToCenter and noPrl

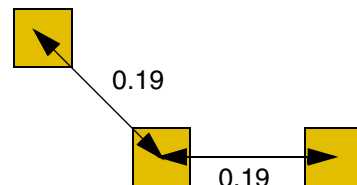
The center-to-center spacing between neighboring via cuts must be at least 0.21 if:

- A via cut has at least two neighboring via cuts less than 0.2 away.
- The via cuts do not have a parallel overlap.

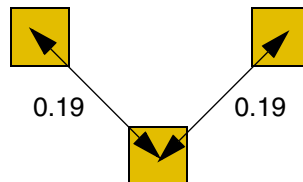
```
spacings(  
  ( viaSpacing "Via1"  
    'centerToCenter  
    'noPrl  
    'numCuts 2  
    'distance 0.2  
    0.21  
  )  
) ;spacings
```



a) The constraint does not apply because there exists only one adjacent via cut.



b) The constraint does not apply because the via cut on the right has parallel run length less than zero with the middle via cut, which means that the via cut on the right is not counted as an adjacent via cut.



c) FAIL. The middle via cut has two adjacent via cuts that have parallel run length less than zero with it. Therefore, the spacing between adjacent via cuts must be at least 0.21.

Example 2: viaSpacing with centerToCenter and exactAligned

The center-to-center spacing between neighboring via cuts must be at least 0.21 if:

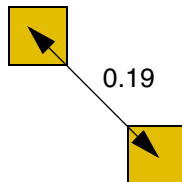
- A via cut has at least two neighboring via cuts less than 0.2 away, and at least one of these neighboring via cuts is not exactly aligned.
- There exist three exactly aligned via cuts, horizontally and/or vertically.

```

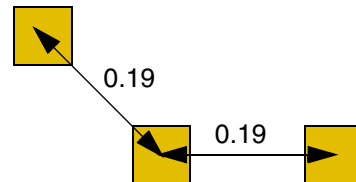
spacings(
  ( viaSpacing "Via1"
    'centerToCenter
    'exactAligned 3
    'numCuts 2
    'distance 0.2
    0.21
  )
) ;spacings

```

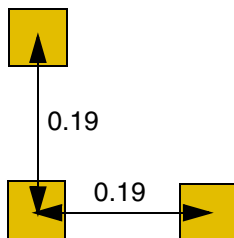
■ Via1



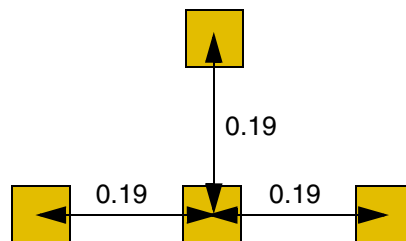
a) The constraint does not apply because there exists only one adjacent via cut.



b) FAIL. The number of adjacent via cuts is two, of which only one is exactly aligned. Therefore, the spacing between adjacent via cuts must be at least 0.21.



c) The constraint does not apply. The number of exactly aligned adjacent via cuts is two.



d) FAIL. There are three exactly aligned adjacent via cuts, two horizontally and one vertically. Therefore, the spacing between adjacent via cuts must be at least 0.21.

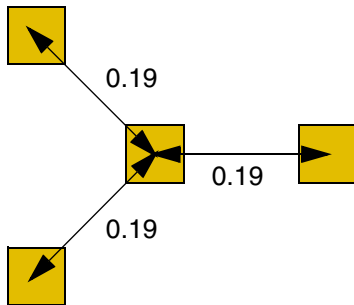
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

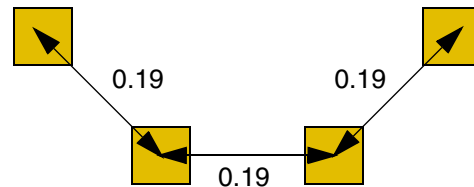
Example 3: viaSpacing with centerToCenter and twoCuts

The center-to-center spacing between two neighboring via cuts must be at least 0.21 if each via cut has exactly two neighboring via cuts less than 0.2 away. Otherwise, there must be at least three adjacent via cuts.

```
spacings(  
  ( viaSpacing "Via1"  
    'centerToCenter  
    'twoCuts 2  
    'numCuts 3  
    'distance 0.2  
    0.21  
  )  
) ;spacings
```



a) FAIL. The middle via cut has three adjacent via cuts. Therefore, the spacing between adjacent via cuts must be at least 0.21.



b) FAIL. Each middle via cut has two adjacent via cuts. Therefore, the spacing between the two middle via cuts must be at least 0.21. The constraint would not apply if 'twoCuts' was not specified.

Virtuoso Technology Data Constraints Reference

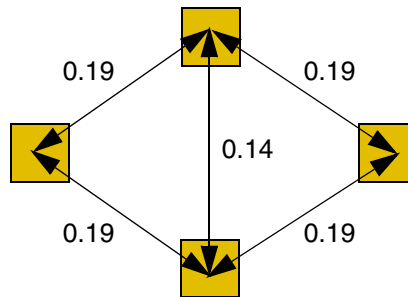
Via Construction Constraints

Example 4: *viaSpacing with centerToCenter, twoCuts, and sameNeighborCuts*

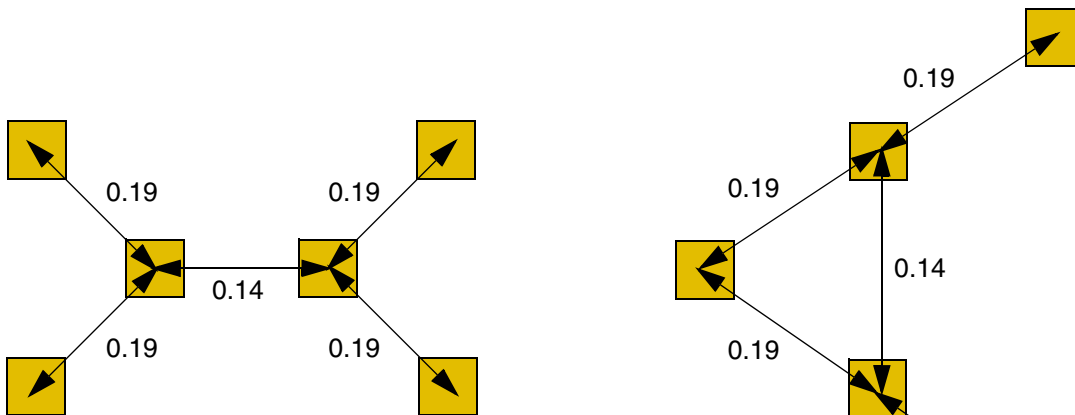
The center-to-center spacing between two neighboring via cuts must be at least 0.21 if each via cut has at least three common neighboring cuts less than 0.2 away. Otherwise, there must be at least four adjacent via cuts.

```
spacings (  
  ( viaSpacing "Via1"  
    'centerToCenter  
    'twoCuts 3 'sameNeighborCuts  
    'numCuts 4  
    'distance 0.2  
    0.21  
  )  
); spacings
```

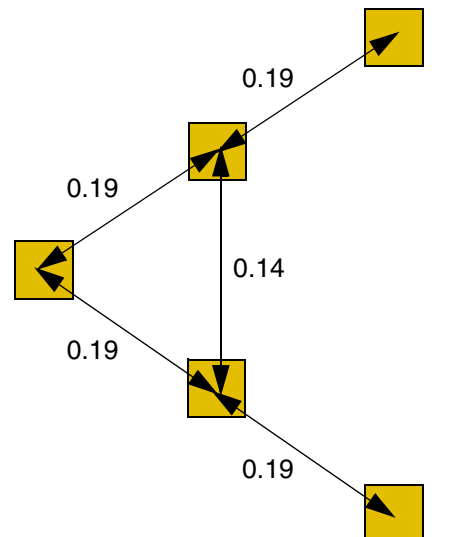
■ Via1



a) FAIL. Both middle via cuts have the same adjacent via cuts as neighbors. Therefore, the spacing between them must be at least 0.21.



b) The constraint does not apply because the two middle via cuts do not have common neighbors. A violation would occur if 'sameNeighborCuts' was not specified.



c) The constraint does not apply. The middle via cuts have only one neighbor in common.

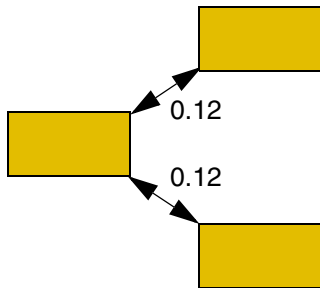
Example 5: viaSpacing with cutClass and noPr1

The edge-to-edge spacing between 0.1x0.2 via cuts must be at least 0.14 if:

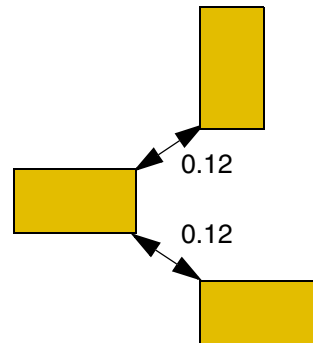
- A via cut has at least two neighboring cuts at a distance less than 0.13 away.
- The via cuts do not have a parallel overlap.

```
spacings (
  ( viaSpacing "Via1"
    'noPr1
    'cutClass (0.1 0.2)
    'numCuts 2
    'distance 0.13
    0.14
  )
) ;spacings
```

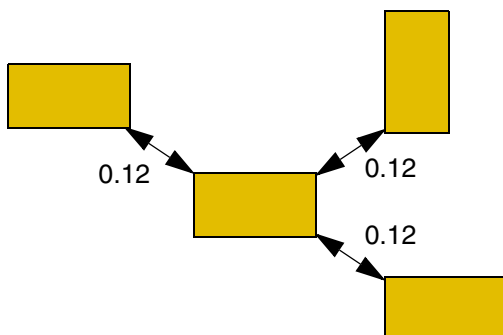
■ Via1



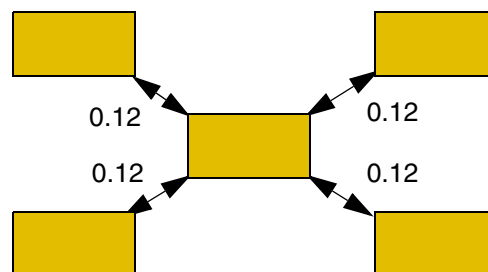
a) FAIL. The middle via cut has two adjacent via cuts that have parallel run length less than zero with it. Therefore, the spacing between them must be at least 0.14.



b) FAIL. The middle via cut has two adjacent via cuts that have parallel run length less than zero with it. Therefore, the spacing between them must be at least 0.14.



c) FAIL. The middle via cut has three adjacent via cuts that have parallel run length less than zero with it. Therefore, the spacing between them must be at least 0.14.



d) FAIL. The middle cut has four adjacent cuts that have parallel run length less than zero with it. Therefore, the spacing between them must be at least 0.14.

Example 6: *viaSpacing* with *cutClass*, *allCuts*, and *noPrl*

The edge-to-edge spacing between a 0.1x0.2 via cut and its neighboring via cuts must be at least 0.14 if:

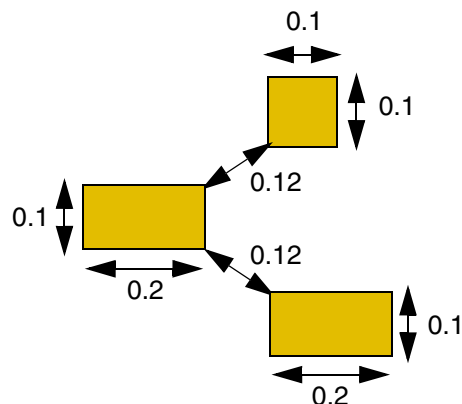
- The via cut has at least two neighboring via cuts, with any dimensions, at a distance less than 0.13 away.
- The via cuts do not have a parallel overlap.

```

spacings(
  ( viaSpacing "Vial"
    'noPrl
    'cutClass (0.1 0.2)
    'allCuts
    'numCuts 2
    'distance 0.13
    0.14
  )
) ;spacings

```

 Via1



b) FAIL. The middle via cut has two adjacent via cuts that have parallel run length less than zero with it. Therefore, the spacing between them must be at least 0.14. The constraint would not apply if 'allCuts' was not specified.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

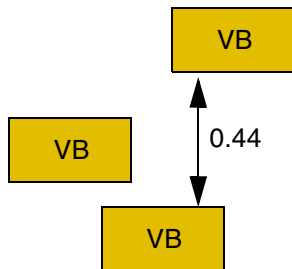
Example 7: *viaSpacing with cutClass and sideParaOverlap*

The edge-to-edge spacing between neighboring via cuts must be at least 0.45 if:

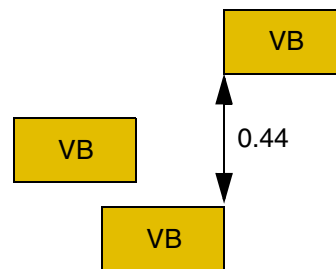
- A VB via cut has at least two neighboring VB via cuts less than 0.45 away.
- The long edges of the via cuts do not have a parallel overlap.

```
spacings(  
  ( viaSpacing "Via1"  
    'cutClass ("VB")  
    'sideParaOverlap  
    'numCuts 2  
    'distance 0.45  
    0.45  
  )  
) ;spacings
```

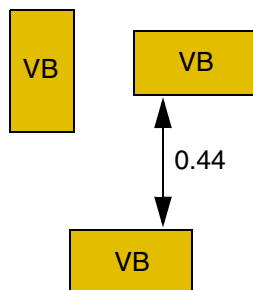
■ Via1



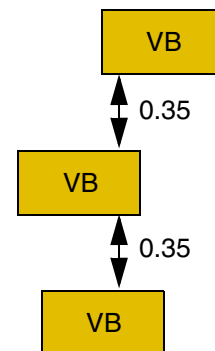
a) FAIL. The long edges of the via cuts have parallel run length greater than zero. Therefore, the spacing between the via cuts must be at least 0.45.



b) The constraint does not apply because the right via cut has zero parallel run length with the middle via cut.



c) The constraint does not apply because the left via cut has an end-to-side parallel edge overlap with the middle via cut. As a result, the left via cut is not counted as an adjacent via cut.



d) FAIL. The side-to-side parallel edge overlap need not be on the same edge. Therefore, the spacing between the via cuts must be at least 0.45.

Example 8: viaSpacing with cutSizeRanges and noPrl

The edge-to-edge spacing between via cuts must be at least 0.14 if:

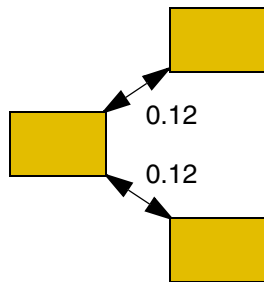
- The first via cut is 0.15 xSpan x 0.1 ySpan and the neighboring via cuts are 0.1 xSpan x 0.1 ySpan.
- The first via cut has at least two neighboring via cuts less than 0.13 away
- The via cuts do not have a parallel overlap.

```

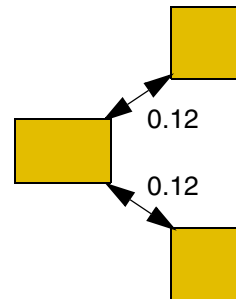
spacings (
  ( viaSpacing "Via1"
    'noPrl
    'cutSizeRanges (0.15 0.1 0.1 0.1)
    'numCuts 2
    'distance 0.13
    0.14
  )
) ;spacings

```

■ Via1



a) Constraint does not apply. The adjacent via cuts must be 0.1 xSpan x 0.1 ySpan.



b) FAIL. The middle 0.15 xSpan x 0.1 ySpan via cut has two adjacent 0.1 xSpan x 0.1 ySpan via cuts that have parallel run length less than zero with it. Therefore, the spacing between them must be at least 0.14.

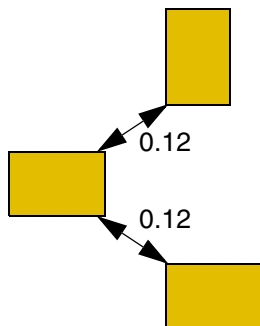
Example 9: viaSpacing with cutSizeRanges and exceptOppositeCornerNeighbors

The edge-to-edge spacing between via cuts must be at least 0.14 for 0.15 xSpan x 0.1 ySpan via cuts if:

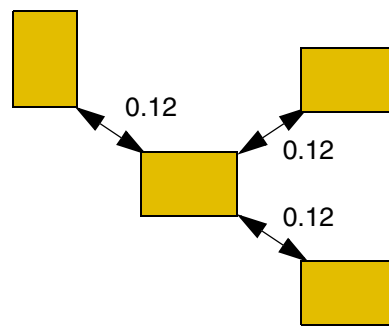
- A via cut has at least two neighboring via cuts less than 0.13 away.
- The adjacent via cuts are not on opposite corners.

```
spacings(
  ( viaSpacing "Via1"
    'cutSizeRanges (0.15 0.1)
    'exceptOppositeCornerNeighbors
    'numCuts 2
    'distance 0.13
    0.14
  )
) ;spacings
```

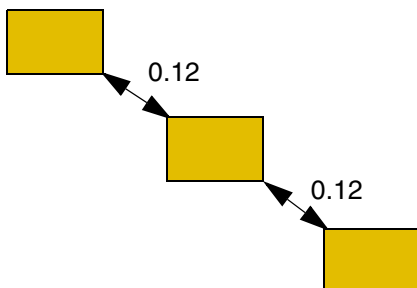
■ Via1



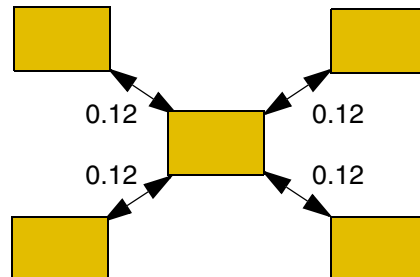
a) The constraint does not apply because only one adjacent via cut exists. The top via cut is not counted as adjacent because its dimensions are 0.1 xSpan x 0.15 ySpan.



b) FAIL. The middle via cut has two adjacent cuts that are not on opposite corners. Therefore, the spacing between them must be at least 0.14.



c) The constraint does not apply because the two adjacent via cuts are on opposite corners.



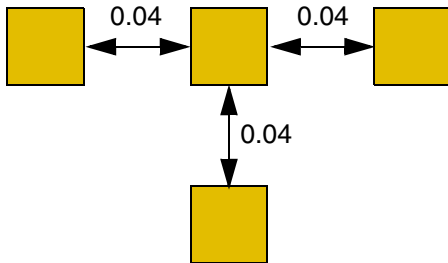
d) The constraint does not apply because all four adjacent via cuts are on opposite corners.

Example 10: viaSpacing with exceptWithin, numCuts, and distance

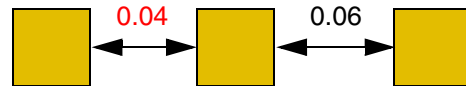
The edge-to-edge spacing between adjacent via cuts must be at least 0.06 if a via cut has two or more neighboring via cuts at a distance less than 0.07 away. The constraint does not apply if all adjacent via cuts are at a distance less than 0.05.

```
spacings (
  ( viaSpacing "Via1"
    'exceptWithin 0.05
    'numCuts 2
    'distance 0.07
    0.06
  )
) ;spacings
```

Via1



a) The constraint does not apply because all adjacent via cuts are at a distance less than 0.05.



b) FAIL. The middle via cut has two neighbors within 0.07, but the spacing between the middle and the left via cuts is only 0.04 (<0.06).

Virtuoso Technology Data Constraints Reference

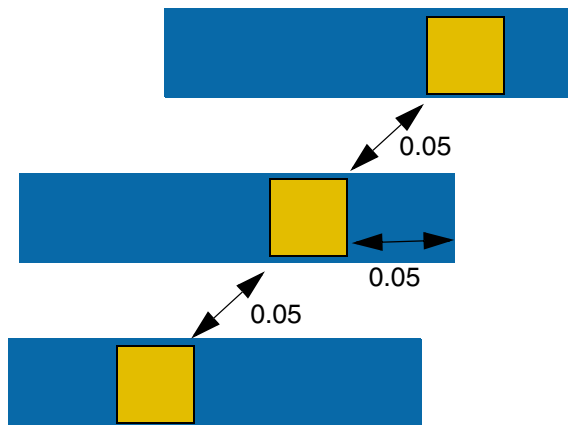
Via Construction Constraints

Example 11: viaSpacing with edgeExtension

The constraint applies only to cuts that have an extension of less than 0.05 in the horizontal direction on layer Metal2.

```
spacings(  
  ( viaSpacing "Via1"  
    'distance 0.07  
    'numCuts 2  
    'edgeExtension 0.05  
    'layer "Metal2"  
    'horizontal  
    0.07
```

■ Via1
■ Metal2



PASS. The middle via has an enclosure of 0.05 in the horizontal direction on Metal2.
FAIL: If 'edgeExtension was not used.

Virtuoso Technology Data Constraints Reference

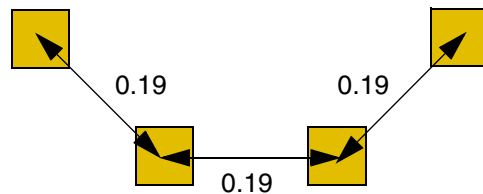
Via Construction Constraints

Example 12: viaSpacing with twoCutsSpacing

The spacing between two neighboring via cuts must be at least 0.2 if each via cut has at least 2 common neighboring cuts that are less than 0.2 away and have spacing less than 0.18. Otherwise, there must be at least 3 adjacent via cuts.

```
spacings(  
  ( viaSpacing "Via1"  
    'numCuts 3  
    'twoCuts 2  
    'twoCutsSpacing 0.18  
    'distance 0.2  
    0.2  
  )  
) ;spacings
```

■ Via1



PASS: The constraint does not apply because the two middle via cuts fulfill the twoCuts condition but do not have a twoCutsSpacing distance of less than 0.18. This would be a violation if twoCutSpacing was not specified.

Virtuoso Technology Data Constraints Reference

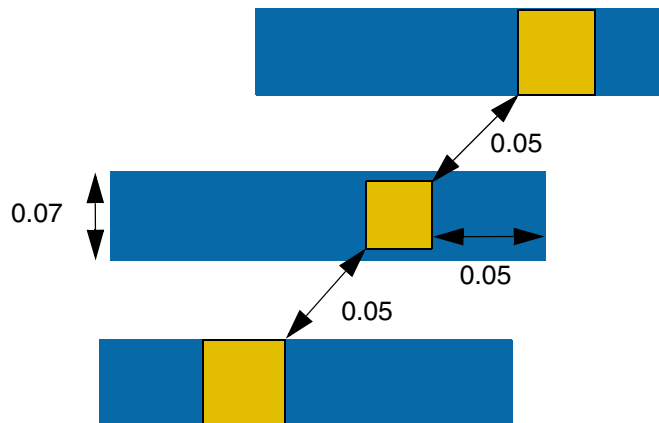
Via Construction Constraints

Example 13: *viaSpacing with width*

The constraint applies only to cuts that have an extension of less than 0.05 in the horizontal direction and width less than or equal to 0.06 on layer `Metal2`.

```
spacings(  
  ( viaSpacing "Via1"  
    'distance 0.07  
    'numCuts 2  
    'edgeExtension 0.05  
    'layer "Metal2"  
    'horizontal  
    'width 0.06  
    0.07  
  )  
) ;spacings
```

■ Via1
■ Metal2



PASS. The middle via has an enclosure of 0.05 in the horizontal direction on `Metal2` and its width is 0.07, which is more than 0.06.

FAIL. If the constraint did not include width.

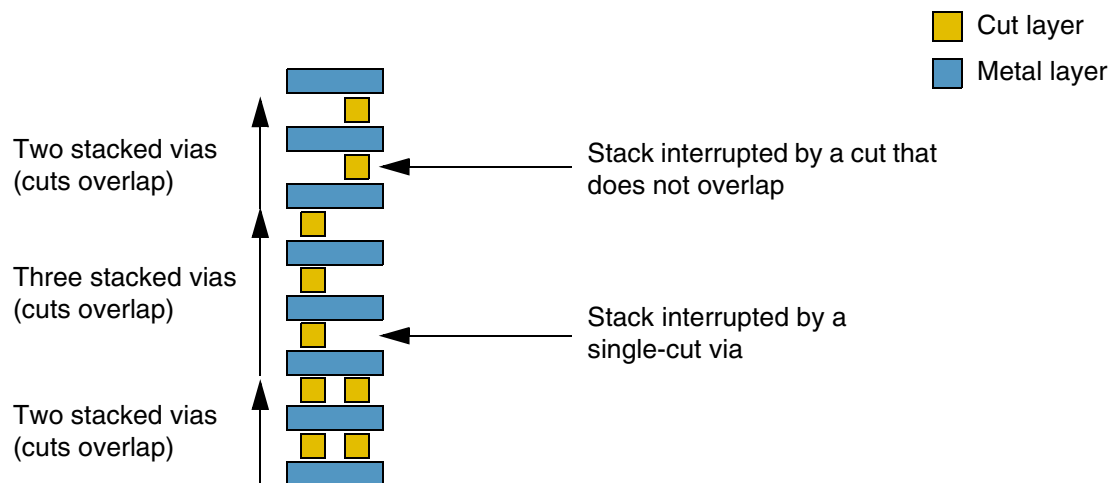
viaStackingLimits

```
viaStackingLimits(  
  ( x_stackLimit  
    ['noSingleCut]  
    ['exceptIndividualArea f_individualArea 'exceptSumArea f_sumArea]  
    ['lowCutSizeWithin]  
    [tx_bottomLayer tx_topLayer]  
  )  
) ;viaStackingLimits
```

Sets the maximum allowed number of continuous stacked vias. A via is considered to be in a stack with another via if a cut in one via partially or completely overlaps a cut in another via. The number of vias in a continuous stack must not exceed the number specified by this constraint.

This constraint must be specified in the `viaStackingLimits` constraint category in the foundry constraint group or in a constraint group applied at the design level. This does not include the global net default constraint groups such as `LEFDefaultRouteSpec` or `virtuosoDefaultSetup`. You can set one `viaStackingLimits` constraint per constraint group; if you set more, the one specified last is applied.

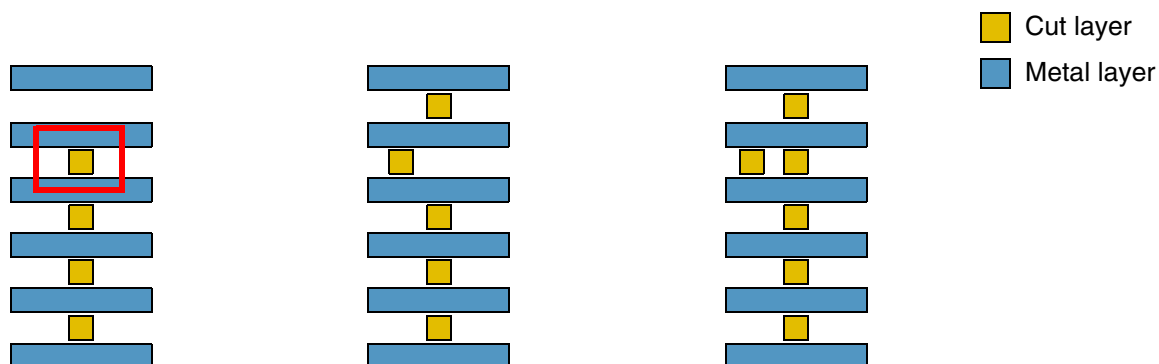
The figure below illustrates via stacks.



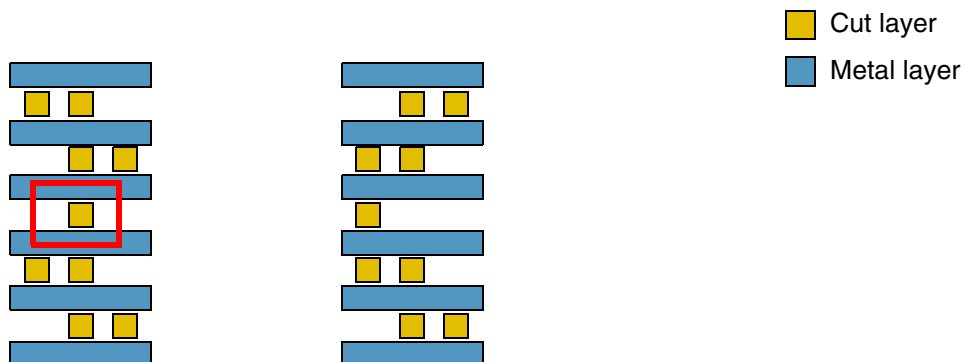
Virtuoso Technology Data Constraints Reference

Via Construction Constraints

In the first example below, if the via stacking limit is set to 3, the leftmost figure violates the constraint because the number of stacked vias is 4 (>3).



In the second example below, if the via stacking limit is set to 3 and 'noSingleCut' is specified, the figure on the left violates the constraint because a single-cut via is not allowed in a stack that is taller than 3.



Values

x_stackLimit

The maximum number of vias that can be stacked.

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Parameters

<code>'noSingleCut</code>	This parameter specifies whether stacked vias can be single-cut or if they must be multiple-cut. ■ If this parameter is not specified, single-cut vias can be stacked up to the maximum specified number. Multiple-cut vias interrupt the stack count, effectively resetting the count to zero. ■ If this parameter is specified, via stacks taller than the stack count must consist of all multiple-cut vias. Type: Boolean
<code>'exceptIndividualArea</code> <i>f_individualArea</i>	The constraint does not apply if the metal shape containing a stacked via has an area greater than or equal to this value on a layer other than the bottommost and topmost metal layers.
<code>'exceptSumArea</code> <i>f_sumArea</i>	The constraint does not apply if the union sum of the metal shapes containing a stacked via on layers other than the bottommost and topmost metal layers is greater than or equal to this value.
<code>'lowCutSizeWithin</code>	It is a stacked via if the upper adjacent layer cut is within the minimum cut size specified on the lower cut layer (as defined in a cutClasses or minWidth constraint). Type: Boolean
<code>tx_bottomLayer</code> <code>tx_topLayer</code>	The two together specify a range of layers to which the constraint applies. If this range is not specified, the constraint applies to all layers. The specified layers must be metal layers. Type: String (layer name) or Integer (layer number)

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Example

The via stack limit is set to 3 for Metal1 through Metal4 layers, and the via stack limit is set to 2 for Metal4 through Metal7 if the vias all have multiple cuts.

```
foundry(  
  viaStackingLimits(  
    ( 3 "Metal1" "Metal4" )  
    ( 2 'noSingleCut "Metal4" "Metal7" )  
  ) ;viaStackingLimits  
) ;foundry
```

Virtuoso Technology Data Constraints Reference

Via Construction Constraints

Constraint Name Mapping for “minEnclosure”, “minExtension”, and “maxEnclosure”

Before IC_6.1.4_ISR1, the issue of inconsistent constraint names between DFII and OA has been raised regarding DFII constraint names, `minEnclosure`, `minExtension`, and `maxEnclosure`. DFII constraint `minEnclosure` was mapped to the OA `oacMinExtension` constraint, DFII constraint `minExtension` was mapped to the OA `oacMinOverlap` constraint, and DFII constraint `maxEnclosure` was mapped to the OA `oacMaxExtension` constraint. This is confusing to DFII applications and users.

Therefore, in IC_6.1.4_ISR1, a new mapping has been introduced to minimize data compatibility issues and address the interoperability problem among the DFII applications as illustrated in the table below.

Old Constraint	New Constraint	OA Constraint Enum
<code>minEnclosure</code>	<code>minExtensionDistance</code>	<code>oacMinExtension</code>
<code>minExtension</code>	<code>minOverlapDistance</code>	<code>oacMinOverlap</code>

Two new DFII constraint names, `minExtensionDistance` and `minOverlapDistance`, are introduced in IC6.1.4_ISR1 and above as illustrated in the table below:

- DFII constraint `minExtensionDistance` is mapped to the OA `oacMinExtension` constraint.
- DFII constraint `minOverlapDistance` is mapped to the OA `oacMinOverlap` constraint.
- DFII constraints `maxEnclosure` and `maxExtension` are mapped to the OA `oacMaxExtension` constraint.

Implementation Specification

DFII constraints `minExtension`, `minEnclosure`, and `maxEnclosure` are deprecated, but ASCII technology file loader and dumper and technology file APIs still accept them and store them in OA as before. However, ASCII technology file loader and tech API `techSetOrderedSpacingRule()` issue the following warnings for `minExtension` and `minEnclosure`. The examples of `minExtension` and `minEnclosure` warning messages are given below:

minExtension

```
(TECH-230023): techSetLayerPairConstraint: "minExtension" constraint
for layer "metal1" and layer "metal2" is deprecated. For backward
compatibility, the constraint will continue to be stored in the
database and will be accessible with APIs. When the technology
database is dumped to ASCII, it will be dumped with the correct
mapping("minOverlapDistance").
```

minEnclosure

```
(TECH-230024): techSetLayerPairConstraint: "minEnclosure" constraint
for layer "metal1" and layer "via" is deprecated. You should use
"minOppExtension" with the same value for two extensions. For
backward compatibility, the constraint will continue to be stored in
the database and will be accessible with APIs. When the technology
database is dumped to ASCII, it will be dumped with the correct
mapping("minExtensionDistance").
```

For DFII `minEnclosure` constraint, it is recommended to replace it with `minOppExtension` with the same extensions. For backward compatibility, this constraint will continue to be stored in the database and will be accessible with APIs.

`techGetOrderedSpacingRule("minEnclosure")` is enhanced to look up `minOppExtension` with two same extensions if the `minOppExtension` constraint exists; otherwise, the API looks up `oacMinExtension` as before.

DFII technology file dumper and API, `techGetOrderSpacingRules()` write out new DFII constraint names. So, the ASCII technology files that are written out by IC6.1.4_ISR1 and above cannot be loaded by IC6.1.4FCS and below. For example, the `minExtensionDistance` constraint cannot be loaded into OA by the IC6.1.4FCS tech reader. Any technology database on disk should not be impacted but may need to be updated to work with the tools, which respect these constraints.

maxEnclosure

Virtuoso Technology Data Constraints Reference

Constraint Name Mapping for “minEnclosure”, “minExtension”, and “maxEnclosure”

For DFII `maxEnclosure` constraint, it is recommended to replace it with `maxExtension`. Although DFII constraint `maxEnclosure` is written out to be `maxExtension`, that should have no impact since the `maxExtension` constraint has already been defined in IC.6.1.4FCS and below.

Virtuoso Technology Data Constraints Reference

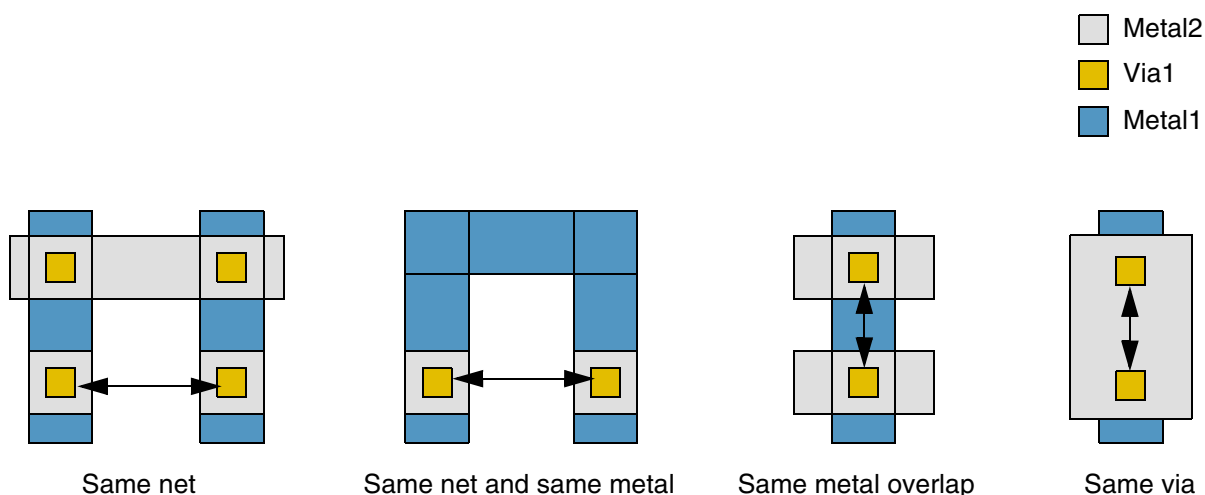
Constraint Name Mapping for “minEnclosure”, “minExtension”, and “maxEnclosure”

Technology File Terms

Connectivity Type

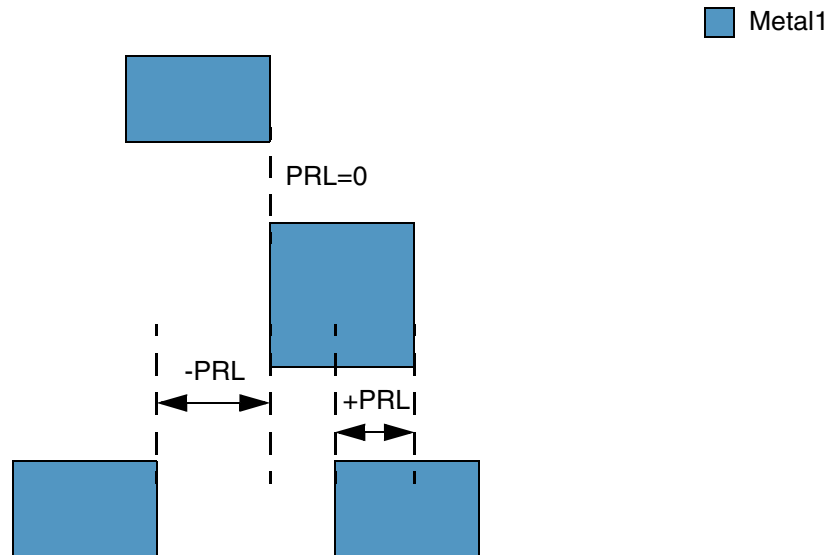
Specifies how shapes are connected. The various connectivity types are illustrated below using a pair of via cuts:

- ☐ Same net: The via cuts are on the same net.
- ☐ Same net and same metal: The via cuts are on a contiguous same-metal shape.
- ☐ Same metal overlap: The via cuts are overlapped by a single metal shape.
- ☐ Same via: The via cuts are overlapped by a single metal shape from above and by another single metal shape from below.



Parallel Run Length

Specifies the overlap between the parallel edges of two neighboring shapes. Parallel run length (PRL) can be positive, negative, or zero.



Range

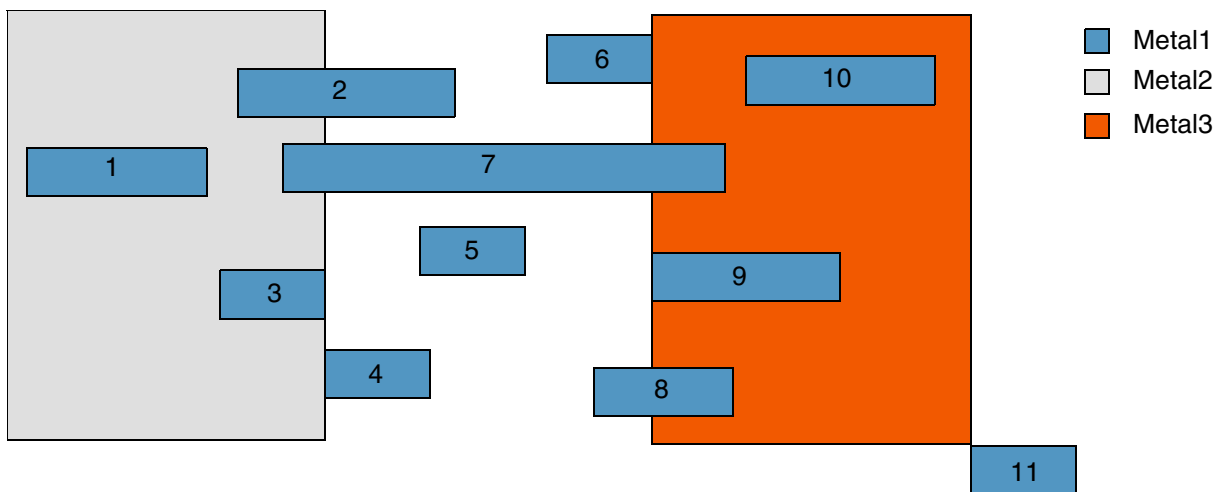
Specifies a set of values that are allowed or exempted. A range can be defined using the following syntax:

- ❑ "[*x_lower* *x_upper*]": The given value must be greater than or equal to *x_lower* and less than or equal to *x_upper*.
- ❑ "(*x_lower* *x_upper*)": The given value must be greater than *x_lower* and less than *x_upper*.
- ❑ "[*x_lower* *x_upper*)": The given value must be greater than or equal to *x_lower* and less than *x_upper*.
- ❑ "(*x_lower* *x_upper*]": The given value must be greater than *x_lower* and less than or equal to *x_upper*.
- ❑ "<*x_value*": The given value must be less than *x_value*.
- ❑ "<=*x_value*": The given value must be less than or equal to *x_value*.
- ❑ ">*x_value*": The given value must be greater than *x_value*.
- ❑ ">=*x_value*": The given value must be greater than or equal to *x_value*.

- ❑ `x_value`: The given value must be equal to `x_value`.

Region-based Rule (One layer)

Determines the shapes to which the constraint applies, based on the presence or absence of one or more layers. For example, in the figure below, Metal1 shapes labeled 2, 7, and 8, which straddle the region boundaries, must pass both `'insideLayers` and `'outsideLayers` rules. Metal1 shapes labeled 1, 3, 9, and 10 must pass only the `'insideLayers` rules for their regions, and Metal1 shapes labeled 4, 5, 6, and 11 must pass only the `'outsideLayers` rules.



For all constraints, except area-specific enclosure constraints, multiple constraints are defined in an AND group. For example, if `minArea` is defined as shown below, the Metal2 `'insideLayers` constraint applies to shapes 1, 2, 3, and 7, and the Metal3 `'insideLayers` constraint applies to shapes 7, 8, 9, and 10, in the figure above. Both these constraints apply to shape 7. The Metal 2 and Metal3 `'outsideLayers` constraint applies to shapes 2, 4, 5, 6, 7, 8, and 11.

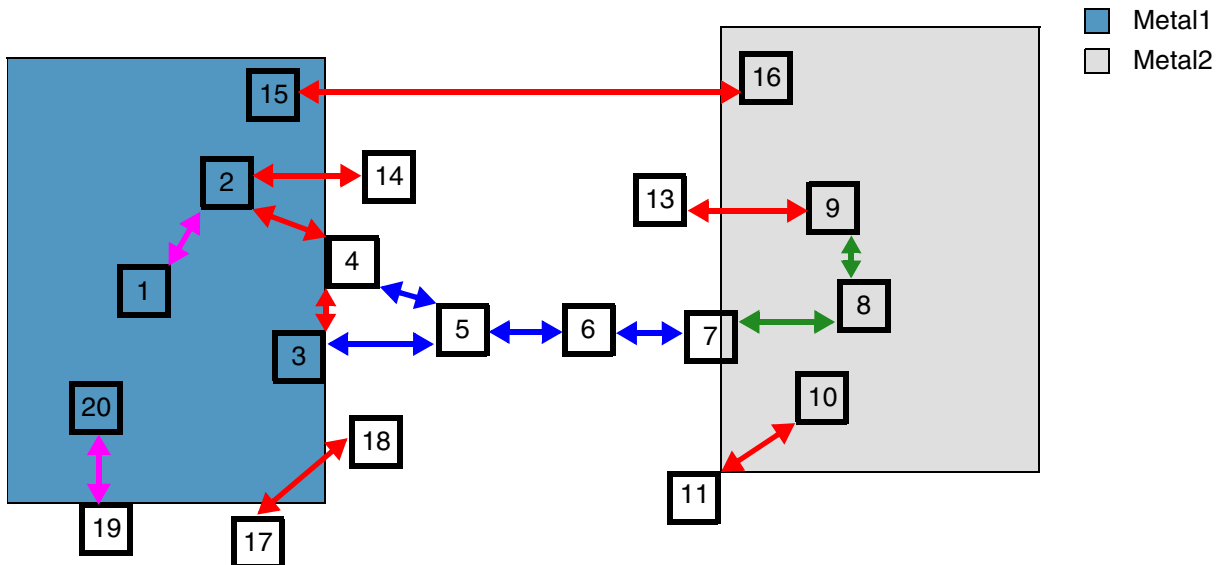
```
constraintGroups
( minAreaGroup nil nil 'and
  spacings(
    ( minArea "Metal1"
      'insideLayers ("Metal2")
      0.03
    )
    ( minArea "Metal1"
      'outsideLayers ("Metal3")
      0.02
    )
    ( minArea "Metal1"
      'outsideLayers ("Metal2" "Metal3")
      0.01
    )
  ) ;spacings
) ;minAreaGroup
```

```
) ;constraintGroups
```

Multiple area-specific enclosure constraints must be defined in an OR group.

Region-based Rule (Two layers)

Determines the shapes to which the constraint applies, based on the presence or absence of one or more layers. For example, in the figure below, the pink arrows show spacing that is entirely inside Metal1 region, the green arrows show spacing that is entirely inside Metal2 region, and the blue arrows show spacing that is entirely outside both Metal1 and Metal2 regions. The red arrows show spacing that is both inside and outside Metal1 and Metal2; both 'insideLayers and 'outsideLayers rules must be applied to these spacings.



When one of the shapes is inside the region and the other abuts the edge of the region, the marker that is generated can be entirely inside the edge of the region (spacing between shapes 19 and 20) or can straddle the edge of the region (spacing between shapes 10 and 11), as shown below. Therefore, pairs such as 10 and 11 must meet both 'insideLayers and 'outsideLayers rules.



Routing Direction

Denotes the direction in which a design is routed—horizontal or vertical. The preferred routing direction is also known as the right-way direction, and the non-preferred routing direction is also known as the wrong-way direction.

Spacing Direction

Denotes the direction in which spacing is measured. The spacing direction is perpendicular to the routing direction.

Span Length

Denotes the distance between two parallel edges, measured perpendicular to the parallel run length.

Via

Comprises a stacked shape on three layers—metal layer / cut layer / metal layer.

Via Cut

Denotes a cut shape associated with a via.

Virtuoso Technology Data Constraints Reference

Technology File Terms

Load CDBA Type Tables

When you load a technology file, you can get some unexpected results if all of the following conditions are true:

- The technology file version is not specified.
- The default value for a constraint is not specified.
- The value table is associated with a constraint other than the following:
 - ❑ `allowedSpacingRanges`
 - ❑ `minCutClassSpacing`
 - ❑ `minSpacing` with index `twoWidth`
 - ❑ `minViaSpacing`
 - ❑ `viaSpacing`

This is because such constraints are interpreted using the "CDBA type" table. A "CDBA type" table is stored as a 1-D or 2-D "OA type" table, but the contents of the specified and missing entries in a "CDBA type" table are populated differently as compared to an "OA type" table. A 1-D table cannot have "missing" entries, but a 2-D table can have "missing" entries.

The syntax of a "CDBA type" table allows you to specify the interpolation type, which is used to fill in the missing entries based on the nearest neighboring entries. For example, in a "snap down" table, if the n^{th} entry is missing, the $n+1^{\text{th}}$ entry is copied to the n^{th} entry. If it is not possible to interpolate the entry (for example, in a table in which the interpolation type is "=="), the entry is set to 0.

Additionally, only one "CDBA style" table can be parsed, adjusted, and stored for a given constraint definition and layer. If multiple tables are defined, only the last one is stored and converted; a warning is not displayed.

Virtuoso Technology Data Constraints Reference

Load CDBA Type Tables

2-D Legacy Table

A 2-D legacy table without a default value, which is considered as a CDBA style table is shown below.

```
constraintGroups (
  ( "foundry" nil
    spacingTables (
      ("minSpacing" "Metal0"
        (( "width" nil nil "length" nil nil ))
        (
          ((0.0 ">=")      ( 0.0 ">="))      0.084
          (( 0.055000 ">=") ( 0.016000 ">=")) 0.070000
          (( 0.116000 ">=") ( 0.130000 ">=")) 0.105000
          (( 0.126000 ">=") ( 0.145000 ">=")) 0.115000
          (( 0.035000 ">=") ( 0.190000 ">=")) 0.105000
          (( 0.060000 ">=") ( 0.200000 ">=")) 0.115000
          (( 0.090000 ">=") ( 0.155000 ">=")) 0.136000
          (( 0.116000 ">=") ( 0.155000 ">=")) 0.150000
        )
      )
    ) ;spacingTables
  ) ;foundry
) ;constraintGroups
```

When such a table is converted to an OA database, the table is populated as shown below:

```
constraintGroups (
  ( "foundry" nil
    spacingTables (
      ("minSpacing" "Metal0"
        (( "width" nil nil "length" nil nil ))
        (
          (0.0 0.0      ) 0.084
          (0.0 0.016    ) 0.084
          (0.0 0.13     ) 0.084
          (0.0 0.145    ) 0.084
          (0.0 0.155    ) 0.084
          (0.0 0.19     ) 0.084
          (0.0 0.2      ) 0.084
          (0.035 0.0    ) 0.084
          (0.035 0.016  ) 0.07
          (0.035 0.13   ) 0.084
        )
      )
    ) ;spacingTables
  ) ;foundry
) ;constraintGroups
```

Virtuoso Technology Data Constraints Reference

Load CDBA Type Tables

(0.035 0.145)	0.084
(0.035 0.155)	0.084
(0.035 0.19)	0.105
(0.035 0.2)	0.084
(0.055 0.0)	0.084
(0.055 0.016)	0.084
(0.055 0.13)	0.105
(0.055 0.145)	0.115
(0.055 0.155)	0.084
(0.055 0.19)	0.084
(0.055 0.2)	0.084
(0.06 0.0)	0.084
(0.06 0.016)	0.084
(0.06 0.13)	0.084
(0.06 0.145)	0.115
(0.06 0.155)	0.084
(0.06 0.19)	0.084
(0.06 0.2)	0.115
(0.09 0.0)	0.084
(0.09 0.016)	0.084
(0.09 0.13)	0.084
(0.09 0.145)	0.084
(0.09 0.155)	0.136
(0.09 0.19)	0.084
(0.09 0.2)	0.084
(0.116 0.0)	0.084
(0.116 0.016)	0.084
(0.116 0.13)	0.084
(0.116 0.145)	0.084
(0.116 0.155)	0.084
(0.116 0.19)	0.084
(0.116 0.2)	0.084
(0.126 0.0)	0.084
(0.126 0.016)	0.084
(0.126 0.13)	0.084
(0.126 0.145)	0.084
(0.126 0.155)	0.084
(0.126 0.19)	0.084
(0.126 0.2)	0.084

)

Virtuoso Technology Data Constraints Reference

Load CDBA Type Tables

```
    ) ;spacingTables
  ) ;foundry
) ;constraintGroups
```

OA Style Table

A table with a default value, which is considered as an OA style table is shown below.

```
constraintGroups (
  ( "foundry" nil
    spacingTables (
      ("minSpacing" "Metal0"
        (( "width" nil nil "length" nil nil ) 0.084)
        (
          (( 0.055000 ">=") ( 0.016000 ">=")) 0.070000
          (( 0.116000 ">=") ( 0.130000 ">=")) 0.105000
          (( 0.126000 ">=") ( 0.145000 ">=")) 0.115000
          (( 0.035000 ">=") ( 0.190000 ">=")) 0.105000
          (( 0.060000 ">=") ( 0.200000 ">=")) 0.115000
          (( 0.090000 ">=") ( 0.155000 ">=")) 0.136000
          (( 0.116000 ">=") ( 0.155000 ">=")) 0.150000
        )
      )
    ) ;spacingTables
  ) ;foundry
) ;constraintGroups
```

For an "OA style" table, the technology file compiler does not support user-defined interpolation types. Therefore, if you specify an interpolation type other than ">=" or "snap down", the technology file reader adjusts the table values while populating the table, so that "snap down" interpolation lookup provides the correct values.

Missing entries are assigned the default value. If the default value is not specified, the missing entries are set to 0. For example, the interpolate type specified in the table above is the compare function. This is ignored and the interpolate type `oacSnapDownInterpolateType` is used.

We recommend that you move away from the CDBA-style tables.