

# **Virtuoso® Technology Data Constraint Reference**

**Product Version ICADV12.3  
March 2017**

© 2016–2017 Cadence Design Systems, Inc. All rights reserved.

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.

**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

---

<u>Preface</u>	13
<u>Scope</u>	13
<u>Licensing Requirements</u>	14
<u>Related Documentation</u>	14
<u>What's New and KPNS</u>	14
<u>Installation, Environment, and Infrastructure</u>	14
<u>Technology Information</u>	14
<u>Virtuoso Tools</u>	14
<u>Relative Object Design and Inherited Connections</u>	15
<u>Additional Learning Resources</u>	15
<u>Video Library</u>	15
<u>Virtuoso Videos Book</u>	15
<u>Rapid Adoption Kits</u>	15
<u>Help and Support Facilities</u>	16
<u>Customer Support</u>	16
<u>Feedback about Documentation</u>	17
<u>Typographic and Syntax Conventions</u>	17
<u>Identifiers Used to Denote Data Types</u>	18
 <u>1</u>	
<u>Constraint Groups and Constraints</u>	21
<u>    Constraint Groups</u>	22
<u>constraintGroups</u>	22
<u>Constraint Groups and Constraint Group Definitions</u>	25
<u>    Constraints</u>	29
<u>Spacing Table Constraints</u>	31
<u>Euclidean and Manhattan Spacing Constraints</u>	32
<u>    Constraint Parameters</u>	33
<u>'hard</u>	33
<u>'soft</u>	33

## Virtuoso Technology Data Constraint Reference

---

<u>'coincidentAllowed</u>	34
<u>'description</u>	34
<u>'ref</u>	35
<u>'manhattan</u>	35
<u>Advanced Nodes Constraints</u>	36
<u>Supported Constraints and Parameters: Advanced Nodes Only</u>	36
<u>Supported Constraints and Parameters: ICADV12.3 Only</u>	41
<b>2</b>	
<u>Antenna Constraints</u>	45
<u>antennaModels</u>	46
<u>maxFloatingArea</u>	52
<b>3</b>	
<u>Area Constraints</u>	59
<u>minArea</u>	60
<u>minAreaEdgeLength</u>	62
<u>minHoleArea</u>	67
<u>minRectArea</u>	69
<b>4</b>	
<u>Density Constraints</u>	73
<u>maxDensity</u>	74
<u>maxDiffDensity</u>	77
<u>minDensity</u>	80
<b>5</b>	
<u>Extension Constraints</u>	83
<u>dummyExtension</u>	85
<u>maxExtension</u>	87
<u>minCenterlineExtension</u>	90
<u>minCornerExtension</u>	94
<u>minEndOfLineEdgeExtension</u>	96

## Virtuoso Technology Data Constraint Reference

---

<u>minEndOfLineExtension</u>	105
<u>minExtensionDistance</u>	109
<u>minExtensionEdge</u>	113
<u>minExtensionOnLongSide (Advanced Nodes Only)</u>	129
<u>minExtensionToCenterLine (Advanced Nodes Only)</u>	132
<u>minExtensionToCorner (Advanced Nodes Only)</u>	137
<u>minGateExtension</u>	149
<u>minInnerVertexProximityExtension (Advanced Nodes Only)</u>	151
<u>minInsideCornerExtension</u>	155
<u>minNeighborExtension (Advanced Nodes Only)</u>	157
<u>minOppEndOfLineExtension</u>	170
<u>minOppExtension</u>	178
<u>minPRBoundaryExtension</u>	200
<u>minProtrusionWidthWithVia (Advanced Nodes Only)</u>	202
<u>minQuadrupleExtension</u>	205
<u>minSideExtension</u>	220
<u>minTouchingDirEnclosure</u>	225
<u>minViaExtension</u>	229
<u>minViaJointExtension</u>	231
<u>minVoltageExtension (ICADV12.3 Only)</u>	234
<u>minWireExtension</u>	236
 <u>6 Length and Width Constraints</u>	
<u>allowedGateLengthRanges</u>	239
<u>allowedGateWidthRanges</u>	241
<u>allowedLengthRanges (Advanced Nodes Only)</u>	243
<u>allowedNeighborWidthRanges (One layer) (Advanced Nodes Only)</u>	245
<u>allowedNeighborWidthRanges (Two layers) (Advanced Nodes Only)</u>	248
<u>allowedNeighborWidthRangesOver (Advanced Nodes Only)</u>	251
<u>allowedPRBoundaryDimensions (Advanced Nodes Only)</u>	254
<u>allowedSpanLengthRanges</u>	257
<u>allowedWidthRanges</u>	259
<u>illegalHGatePattern</u>	262
<u>maxDiagonalEdgeLength</u>	270
	272

<u>maxDiffusionLength</u>	273
<u>maxLength</u>	275
<u>maxNumMinEdges</u>	276
<u>maxPolyLength</u>	278
<u>maxTouchingDirectionLength</u>	280
<u>maxWidth</u>	282
<u>minChamferLength</u>	283
<u>minDiagonalEdgeLength</u>	288
<u>minDiagonalWidth</u>	289
<u>minEdgeAdjacentDistance</u>	290
<u>minEdgeAdjacentLength</u>	292
<u>minEndOfLineAdjacentToStep</u>	299
<u>minInsideCornerEdgeLength</u>	305
<u>minLength</u>	307
<u>minOutsideCornerEdgeLength</u>	308
<u>minPerimeter</u>	310
<u>minProtrusionWidthLength</u>	311
<u>minSize</u>	314
<u>minStepEdgeLength</u>	316
<u>minWidth</u>	320
<u>protrusionWidth</u>	323

## 7

<u>Miscellaneous Constraints</u>	325
<u>allowedShapeAngles</u>	327
<u>allowedSnapPatternDefs (ICADV12.3 Only)</u>	329
<u>allowedWireTypes (ICADV12.3 Only)</u>	330
<u>diagonalShapesAllowed</u>	331
<u>edgeMustCoincide (Advanced Nodes Only)</u>	332
<u>edgeMustOverlap (Advanced Nodes Only)</u>	334
<u>errorLayer</u>	338
<u>gateOrientation</u>	339
<u>maxACCCurrentDensity</u>	341
<u>maxNumCorners (Advanced Nodes Only)</u>	345
<u>minOneDArrayStructure</u>	346

<u>minStitchOverlap</u> (Advanced Nodes Only) .....	349
<u>msCrossTieLayers</u> .....	353
<u>msLayerCrossTieInterval</u> .....	354
<u>msShieldingLimit</u> .....	355
<u>msShieldStyle</u> .....	356
<u>multiMaskCheck</u> (ICADV12.3 Only) .....	358
<u>rectShapeDir</u> .....	359
<u>sameMaskOnLayer</u> (ICADV12.3 Only) .....	368
<u>trimMetalTrack</u> (ICADV12.3 Only) .....	369
<u>trimShape</u> (ICADV12.3 Only) .....	371
<u>validCutClass</u> .....	377
<u>vertexInsideForbidden</u> (ICADV12.3 Only) .....	379
 <u>8</u>	
<u>NumCut Constraints</u> .....	381
<u>maxViaClusterNumCuts</u> .....	382
<u>minNumCut</u> .....	388
<u>minProtrusionNumCut</u> .....	395
 <u>9</u>	
<u>Overlap Constraints</u> .....	401
<u>minDirectionalOverlap</u> (Advanced Nodes Only) .....	402
<u>minInsideCornerOverlap</u> .....	405
<u>minOverlapDistance</u> .....	407
<u>minWireOverlap</u> (Advanced Nodes Only) .....	409
 <u>10</u>	
<u>Placement and Alignment Constraints</u> .....	413
<u>allowedWidthSpacingPatternGroups</u> (ICADV12.3 Only) .....	414
<u>allowedWidthSpacingPatterns</u> (ICADV12.3 Only) .....	415
<u>defaultActiveWidthSpacingPattern</u> (ICADV 12.3 Only) .....	416
<u>horizontalOffset</u> .....	417
<u>horizontalPitch</u> .....	418
<u>keepPRBoundarySharedEdges</u> .....	419

<u>keepSharedEdges</u>	421
<u>snapGridHorizontal</u>	423
<u>snapGridVertical</u>	424
<u>verticalOffset</u>	425
<u>verticalPitch</u>	426
<b>11</b>	
<b>Routing Constraints</b>	427
<u>clusterDistance</u>	429
<u>excludeLPPs</u>	430
<u>horizontalOffset</u>	431
<u>horizontalPitch (One layer)</u>	432
<u>ladderFrequency</u>	434
<u>ladderOffset</u>	436
<u>ladderStyle</u>	437
<u>layerMaskShiftAllowed (ICADV12.3 Only)</u>	439
<u>leftDiagOffset</u>	440
<u>leftDiagPitch</u>	442
<u>maxRoutingDistance</u>	444
<u>numStrands</u>	445
<u>orthogonalSnappingLayer (ICADV12.3 Only)</u>	448
<u>orthogonalWSPGrid (ICADV12.3 Only)</u>	451
<u>rectangularGapMinSpacing (ICADV12.3 Only)</u>	453
<u>rightDiagOffset</u>	457
<u>rightDiagPitch</u>	458
<u>routingDirections</u>	459
<u>strandSpacing</u>	461
<u>strandWidth</u>	462
<u>trackPattern</u>	463
<u>taperHalo</u>	464
<u>validLayers</u>	465
<u>validPurposes</u>	466
<u>validVias</u>	468
<u>verticalOffset</u>	471
<u>verticalPitch (One layer)</u>	472

### 12

<u>Spacing Constraints (One Layer)</u> . . . . .	475
<u>allowedBetweenNeighborWidthRanges (ICADV12.3 Only)</u> . . . . .	478
<u>allowedSpacingRanges (One layer)</u> . . . . .	482
<u>cornerFillSpacing</u> . . . . .	492
<u>endOfLineKeepout</u> . . . . .	495
<u>forbiddenEdgePitchRange (Advanced Nodes Only)</u> . . . . .	507
<u>forbiddenProximitySpacing (Advanced Nodes Only)</u> . . . . .	523
<u>gateSpacingRanges</u> . . . . .	527
<u>maxFilling</u> . . . . .	529
<u>maxTapSpacing</u> . . . . .	531
<u>minAdjacentFourViaSpacing (ICADV12.3 Only)</u> . . . . .	533
<u>minCenterToCenterSpacing</u> . . . . .	536
<u>minClusterSpacing (One layer) (Advanced Nodes Only)</u> . . . . .	538
<u>minConcaveCornerSpacing (ICADV12.3 Only)</u> . . . . .	549
<u>minConvexCornerSpacing</u> . . . . .	553
<u>minCornerSpacing (One layer) (Advanced Nodes Only)</u> . . . . .	559
<u>minCornerToCornerDistance</u> . . . . .	570
<u>minDiagonalSpacing</u> . . . . .	571
<u>minDiffNetSpacing</u> . . . . .	574
<u>minEdgeLengthSpacing (Advanced Nodes Only)</u> . . . . .	576
<u>minEndOfLineExtensionSpacing</u> . . . . .	578
<u>minEndOfLinePerpSpacing</u> . . . . .	585
<u>minEndOfLineSpacing</u> . . . . .	587
<u>minEndOfLineToConcaveCornerSpacing</u> . . . . .	622
<u>minEndOfLineToNotchSpacing (Advanced Nodes Only)</u> . . . . .	625
<u>minEndOfNotchSpacing</u> . . . . .	627
<u>minExtensionSpacing</u> . . . . .	630
<u>minFillToFillSpacing</u> . . . . .	633
<u>minFillToShapeSpacing</u> . . . . .	635
<u>minFiveWiresEndOfLineSpacing (ICADV12.3 Only)</u> . . . . .	637
<u>minHoleWidth</u> . . . . .	641
<u>minInfluenceSpacing</u> . . . . .	643
<u>minJointCornerSpacing</u> . . . . .	646
<u>minLargeNeighborViaArrayCutSpacing</u> . . . . .	648

<u>minNotchSpacing</u>	.....	651
<u>minNotchSpanSpacing</u>	.....	660
<u>minOppSpanSpacing</u>	.....	663
<u>minOrthogonalSpacing (ICADV12.3 Only)</u>	.....	676
<u>minOuterVertexSpacing</u>	.....	678
<u>minParallelSpanSpacing</u>	.....	680
<u>minPRBoundaryExteriorHalo</u>	.....	685
<u>minPRBoundaryInteriorHalo</u>	.....	687
<u>minPrlTwoSidesSpacing (ICADV12.3 Only)</u>	.....	690
<u>minProtrusionSpacing</u>	.....	693
<u>minSameNetSpacing (One layer)</u>	.....	701
<u>minSideSpacing (One layer) (Advanced Nodes Only)</u>	.....	703
<u>minSpacing (One layer)</u>	.....	710
<u>minSpanLengthSpacing (Advanced Nodes Only)</u>	.....	731
<u>minStubInfluenceSpacing</u>	.....	754
<u>minVoltageSpacing (One layer)</u>	.....	757
<u>oneSideSpacing (ICADV12.3 Only)</u>	.....	760
<u>pgViaTrack (Advanced Nodes Only)</u>	.....	762
<u>shapeRequiredBetweenSpacing (Advanced Nodes Only)</u>	.....	764
<u>snapPatternDefOffset (ICADV12.3 Only)</u>	.....	766
<u>trimMinAdjacentSpacing (ICADV12.3 Only)</u>	.....	767
<u>trimMinSpacing (One layer) (ICADV12.3 Only)</u>	.....	769
<u>viaKeepoutZone (ICADV12.3 Only)</u>	.....	778

## 13

<u>Spacing Constraints (Two Layers)</u>	.....	783
<u>allowedSpacingRanges (Two layers)</u>	.....	784
<u>maxSpacing</u>	.....	787
<u>minCenterLineSpacing (Two layers) (Advanced Nodes Only)</u>	.....	788
<u>minClusterSpacing (Two layers) (Advanced Nodes Only)</u>	.....	790
<u>minCornerSpacing (Two layers) (Advanced Nodes Only)</u>	.....	797
<u>minCutRoutingSpacing (Two layers)</u>	.....	799
<u>minInnerVertexSpacing (Two layers)</u>	.....	808
<u>minInnerVertexSpacing (Three layers)</u>	.....	810
<u>minNeighboringShapesSpacing (Advanced Nodes Only)</u>	.....	812

<u>minSameNetSpacing (Two layers)</u>	816
<u>minSideSpacing (Two layers) (Advanced Nodes Only)</u>	818
<u>minSpacing (Two layers)</u>	825
<u>minSpacingOver</u>	834
<u>minTouchingDirSpacing</u>	836
<u>minVoltageSpacing (Two layers)</u>	840
<u>shapeRequiredSpacing</u>	844
<u>trimMinSpacing (Two layers) (ICADV12.3 Only)</u>	848

## 14

### Via Construction Constraints ..... 851

<u>allowedCutClass (Advanced Nodes Only)</u>	853
<u>definedCutClassesOnly</u>	858
<u>forbiddenCutClassSpacingRange (Advanced Nodes Only)</u>	859
<u>largeRectViaArrayAllowed</u>	865
<u>minCutClassSpacing (One layer)</u>	867
<u>minCutClassSpacing (Two layers)</u>	886
<u>minEndOfLineCutSpacing</u>	899
<u>minLargeViaArrayCutSpacing</u>	905
<u>minLargeViaArraySpacing</u>	908
<u>minLargeViaArrayWidth</u>	913
<u>minNeighborViaSpacing</u>	915
<u>minOrthogonalViaSpacing</u>	917
<u>minParallelViaSpacing (One layer)</u>	919
<u>minParallelViaSpacing (Two layers plus metal)</u>	921
<u>minParallelWithinViaSpacing</u>	923
<u>minSameMetalSharedEdgeViaSpacing</u>	929
<u>minViaSpacing (One layer)</u>	934
<u>minViaSpacing (Two layers)</u>	945
<u>redundantViaSetback</u>	952
<u>stackable</u>	954
<u>viaEdgeType</u>	955
<u>viaRequiredWithin (Advanced Nodes Only)</u>	960
<u>viaSpacing</u>	962
<u>viaStackingLimits</u>	980

**A**

<u>Constraint Name Mapping for “minEnclosure”,</u>	
<u>“minExtension”, and “maxEnclosure”</u>	..... 985
<u>Introduction</u>	..... 985
<u>Implementation Specification</u>	..... 986

**B**

<u>Glossary</u>	..... 989
-----------------	-----------

**C**

<u>Loading "CDBA Type" Tables</u>	..... 995
-----------------------------------	-----------

# Preface

---

This manual provides detailed information about the syntax of the statements you code into your technology file and display resource file to define the constraints and display attributes for your design session.

This preface contains the following topics:

- [Scope](#)
- [Licensing Requirements](#)
- [Related Documentation](#)
- [Additional Learning Resources](#)
- [Customer Support](#)
- [Feedback about Documentation](#)
- [Typographic and Syntax Conventions](#)
- [Identifiers Used to Denote Data Types](#)

## Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.7) and advanced node (for example, ICADV12.3) releases.

- Features supported only in mature node releases are identified using the **(IC6.1.7 Only)** label.
- Features supported only in advanced node releases are identified using the **(Advanced Nodes Only)** label and require an advanced node license.
- There is also an exclusive set of advanced node features that is supported only in ICADV12.3 and which requires the Virtuoso Advanced Node Option for Layout (95511) license. These features are identified using the **(ICADV12.3 Only)** label.

## Licensing Requirements

For information about licensing in the Virtuoso design environment, see [Virtuoso Software Licensing and Configuration User Guide](#).

## Related Documentation

### What's New and KPNS

- [Virtuoso Technology Data What's New](#)
- [Virtuoso Technology Data Known Problems and Solutions](#)

### Installation, Environment, and Infrastructure

- [Cadence Installation Guide](#)
- [Virtuoso Design Environment User Guide](#)
- [Virtuoso Design Environment SKILL Reference](#)
- [Cadence Application Infrastructure User Guide](#)

### Technology Information

- [Virtuoso Technology Data ASCII Files Reference](#)
- [Virtuoso Technology Data User Guide](#)
- [Virtuoso Technology Data SKILL Reference](#)

### Virtuoso Tools

- [Virtuoso Layout Suite L User Guide](#)
- [Virtuoso Design Rule Driven Editing User Guide](#)
- "Interactive Wire Editing" in the [Virtuoso Space-based Router User Guide](#)
- [Virtuoso Custom Digital Placer User Guide](#)
- [Virtuoso Parameterized Cell Reference](#)

- [Component Description Format User Guide](#)
- [Virtuoso Space-based Router User Guide](#)
- [Design Data Translator's Reference](#)
- [Virtuoso Layout Suite SKILL Reference](#)

## Relative Object Design and Inherited Connections

- [Virtuoso Relative Object Design User Guide](#)
- [Virtuoso Relative Object Design SKILL Reference](#)
- [Virtuoso Schematic Editor L User Guide](#)

## Additional Learning Resources

### Video Library

The [Video Library](#) on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

### Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see [Virtuoso Videos](#).

### Rapid Adoption Kits

Cadence provides a number of [Rapid Adoption Kits](#) that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

To explore the full range of training courses provided by Cadence in your region, visit [Cadence Training](#) or write to [training\\_enroll@cadence.com](mailto:training_enroll@cadence.com).

**Note:** The links in this section open in a separate web browser window when clicked in Cadence Help.

## Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see [Getting Help](#) in *Virtuoso Design Environment User Guide*.

## Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit <https://www.cadence.com/support>.

- Log on to Cadence Online Support

Customers with a maintenance contract with Cadence can obtain the latest information about various tools at <https://support.cadence.com>.

## Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support [Product Manuals](#) page, select the required product and submit your feedback by using the *Provide Feedback* box.

## Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

<i>text</i>	Indicates names of manuals, menu commands, buttons, and fields.
<i>text</i>	Indicates text that you must type exactly as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
<i>z_argument</i>	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, <i>z_</i> ) indicates the data type the argument can accept and must not be typed.
	Separates a choice of options.
{ }	Encloses a list of choices, separated by vertical bars, from which you <b>must</b> choose one.
[ ]	Encloses an optional argument or a list of choices separated by vertical bars, from which you <b>may</b> choose one.
[ ?argName <i>t_arg</i> ]	Denotes a <i>key argument</i> . The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.

## Virtuoso Technology Data Constraint Reference

### Preface

---

...	Indicates that you can repeat the previous argument.
	Used with brackets to indicate that you can specify zero or more arguments.
	Used without brackets to indicate that you must specify at least one argument.
, . . .	Indicates that multiple arguments must be separated by commas.
=>	Indicates the values returned by a Cadence® SKILL® language function.
/	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash ( \ ) indicates that the current line continues on to the next line. If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash ( \ ) indicates that the current line continues on to the next line.

## Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, *t* is the data type in *t\_viewName*. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

**Note:** All values are specified in user units unless otherwise noted.

---

Prefix	Internal Name	Data Type
<i>a</i>	array	array
<i>A</i>	amsobject	AMS object
<i>b</i>	ddUserType	DDPI object
<i>B</i>	ddCatUserType	DDPI category object
<i>C</i>	opfcontext	OPF context

## Virtuoso Technology Data Constraint Reference

### Preface

---

<b>Prefix</b>	<b>Internal Name</b>	<b>Data Type</b>
<i>d</i>	dbobject	Cadence database object (CDBA)
<i>e</i>	envobj	environment
<i>f</i>	flonum	floating-point number
<i>F</i>	opffile	OPF file ID
<i>g</i>	general	any data type
<i>G</i>	gdmSpecIIUserType	generic design management (GDM) spec object
<i>h</i>	hdbobject	hierarchical database configuration object
<i>I</i>	dbgenobject	CDB generator object
<i>K</i>	mapiobject	MAPI object
<i>l</i>	list	linked list
<i>L</i>	tc	Technology file time stamp
<i>m</i>	nmpIIUserType	nmpII user type
<i>M</i>	cdsEvalObject	cdsEvalObject
<i>n</i>	number	integer or floating-point number
<i>o</i>	userType	user-defined type (other)
<i>p</i>	port	I/O port
<i>q</i>	gdmspecListIIUserType	gdm spec list
<i>r</i>	defstruct	defstruct
<i>R</i>	rodObj	relative object design (ROD) object
<i>s</i>	symbol	symbol
<i>S</i>	stringSymbol	symbol or character string
<i>t</i>	string	character string (text)
<i>T</i>	txobject	transient object
<i>u</i>	function	function object, either the name of a function (symbol) or a lambda function body (list)
<i>U</i>	funobj	function object
<i>v</i>	hdbpath	hdbpath
<i>w</i>	wtype	window type

## Virtuoso Technology Data Constraint Reference

### Preface

---

Prefix	Internal Name	Data Type
<i>sw</i>	swtype	subtype session window
<i>dw</i>	dwtype	subtype dockable window
<i>x</i>	integer	integer number
<i>y</i>	binary	binary function
<i>&amp;</i>	pointer	pointer type

---

For more information, see [\*Cadence SKILL Language User Guide\*](#).

---

# Constraint Groups and Constraints

---

This chapter includes the following topics:

- [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.

To download a spreadsheet with constraint-to-application mapping information, click [Constraints Map v2.1](#) (286 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 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 Constraint 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 Groups and 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 Constraint 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.

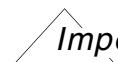
**Example:**

```
; ( 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



colorControl is similar to other constraint categories such as spacings, orderedSpacings, and routingGrids.

**Example**

```
colorControl(
    ( integrationColorModel "locked")
) ;colorControl
```

*l\_constraint*      The constraint specification; as defined in [Constraints](#).

## Constraint Groups and Constraint Group Definitions

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

Constraint groups are categorized as follows:

- **foundry**

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.

- **Application-specific constraint groups**

An application-specific constraint group influences the behavior of a particular application. Examples of predefined application-specific constraint groups include:

- `virtuosoDefaultExtractorSetup`: Is used by default by the extractor for connectivity extraction.
- `virtuosoDefaultSetup`: Is used by default for interactive and automatic routing.
- `LEFDefaultRouteSpec`: Stores routing information, such as information about layers and vias.
- `LEFSpecialRouteSpec`: Stores information about power routing vias.

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.

## Virtuoso Technology Data Constraint Reference

### Constraint Groups and Constraints

---

#### ■ 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.

A constraint group definition is used to identify the type of objects to which the constraint group applies. Valid values for constraint group definitions include:

- `LEFDefaultRouteSpec`
- `LEFSpecialRouteSpec`
- `taper`
- `inputTaper`
- `outputTaper`
- `shielding`
- `virtuosoExtractorSetup`
- `virtuosoMPTSetup`

These constraint group definitions provide constraint group types for specific design functions; for example, `taper` is for wire tapering, `virtuosoExtractorSetup` is for extractor-related constraints, and `virtuosoMPTSetup` is for color-control-related 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 "Metal12"
    '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
    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 Constraint 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 <a href="#">Constraint Parameters</a> .
<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 "Metall1" "Via1"
        'cutDistance 0.1      ;first parameter
        'noSharedEdge       ;second parameter
        (2.1 0.7)          ;constraint value
        'soft
        'ref "M1.EX.1"
        'description "Metall1 extension past Via1"
    )
) ;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. 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 0 for the missing table entries.

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

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

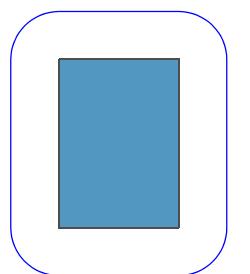
The first value in the table that matches a query during lookup is returned.

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

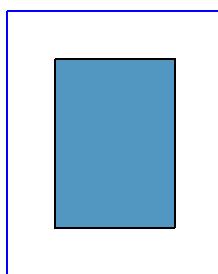
## Euclidean and Manhattan Spacing Constraints

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.



Euclidean spacing halo



Manhattan spacing halo

- █ Layer
- █ Spacing halo circumscribing the area to which the spacing constraint applies

## 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.

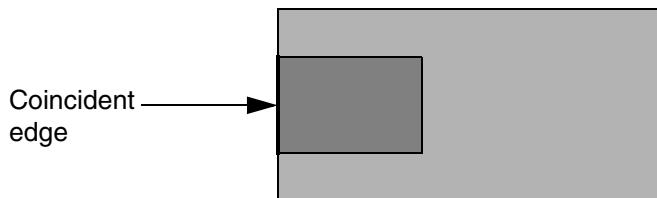
#### Example

```
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
```

## '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

<i>t_ref</i>	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:

- **Advanced Nodes Only**

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

- **ICADV12.3 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, only the parameters are advanced node and 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: Advanced Nodes Only

Constraint Name	Parameters
<code>allowedCutClass</code>	
<code>allowedLengthRanges</code>	
<code>allowedNeighborWidthRanges</code> <code>(One layer)</code>	
<code>allowedNeighborWidthRanges</code> <code>(Two layers)</code>	
<code>allowedNeighborWidthRangesOver</code>	
<code>allowedPRBoundaryDimensions</code>	

## Virtuoso Technology Data Constraint Reference

### Constraint Groups and Constraints

Constraint Name	Parameters
<u>allowedSpacingRanges (One layer)</u>	widthRanges, exceptOverLayer, overLayerPrl, overLayerWidthRanges, exceptNumShapes, numShapeDistance, numShapeWidthRanges, numShapeMaxWidth, stepSize, stepRange, overLayer
<u>allowedWidthRanges</u>	measurementHorizontal, 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>maxNumCorners</u>	
<u>minCenterLineSpacing (Two layers)</u>	
<u>minClusterSpacing (One layer)</u>	
<u>minClusterSpacing (Two layers)</u>	
<u>minCornerSpacing (One layer)</u>	
<u>minCornerSpacing (Two layers)</u>	
<u>minCutClassSpacing (One layer)</u>	bothNegativePrls, nonCutClassEdgeSpacing, sameMask
<u>minCutClassSpacing (Two layers)</u>	nonZeroEnclosure, nonCutClassEdgeSpacing, cutClassSizeBy

## Virtuoso Technology Data Constraint Reference

### Constraint Groups and Constraints

---

<b>Constraint Name</b>	<b>Parameters</b>
<u><a href="#">minCutRoutingSpacing (Two layers)</a></u>	enclosingLayerWidth, enclosingLayer, maskOverlap, parallelExtension, prl, parallelEdgeWithin, otherExtension, anyOppositeExtension, viaEdgeType
<u><a href="#">minDirectionalOverlap</a></u>	
<u><a href="#">minEdgeAdjacentLength</a></u>	concaveCorner
<u><a href="#">minEndOfLineAdjacentToStep</a></u>	adjacentLength, concaveCorner
<u><a href="#">minEndOfLineExtensionSpacing</a></u>	negativePRL, sameMask
<u><a href="#">minEndOfLineSpacing</a></u>	exactEolWidth, horizontal, vertical, extendBy, sizeBy, widthRanges, sameMask, diffMask, sameNet, sameMetal, exceptExactAligned, bothWires, wrongDirSpace
<u><a href="#">minEndOfLineToConcaveCornerSpacing</a></u>	dualAdjacentLength
<u><a href="#">minEndOfLineToNotchSpacing</a></u>	
<u><a href="#">minExtensionDistance</a></u>	horizontal, vertical, cutClass, exceptEdgeLengthRange
<u><a href="#">minExtensionEdge</a></u>	exceptConcaveCorner, twoSides, horizontal, vertical
<u><a href="#">minExtensionOnLongSide</a></u>	
<u><a href="#">minExtensionToCenterLine</a></u>	
<u><a href="#">minExtensionToCorner</a></u>	
<u><a href="#">minInnerVertexProximityExtension</a></u>	
<u><a href="#">minJointCornerSpacing</a></u>	sameMask
<u><a href="#">minLargeViaArrayCutSpacing</a></u>	maxNumCuts
<u><a href="#">minNeighborExtension</a></u>	
<u><a href="#">minNeighboringShapesSpacing</a></u>	
<u><a href="#">minNotchSpacing</a></u>	notchWidth

**Virtuoso Technology Data Constraint Reference**  
Constraint Groups and Constraints

---

Constraint Name	Parameters
<u>minNumCut</u>	sameMetalOverlap
<u>minOppExtension</u>	horizontal, vertical, includeAbutted, stepSizePair, exactPrl, hasExactPrl
<u>minParallelWithinViaSpacing</u>	cutClass, longEdgeOnly
<u>minPRBoundaryInteriorHalo</u>	horizontal, vertical
<u>minProtrusionSpacing</u>	excludeSpacing
<u>minProtrusionWidthLength</u>	enclosedWidth
<u>minProtrusionWidthWithVia</u>	
<u>minRectArea</u>	maxWidth
<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
<u>minWireOverlap</u>	
<u>pgViaTrack</u>	

## Virtuoso Technology Data Constraint Reference

### Constraint Groups and Constraints

---

Constraint Name	Parameters
<u>rectShapeDir</u>	extendBy, exceptViaLayer, exceptViaSize, exceptExactSize, exceptEdgeLength, twoSides, width, widthRanges
<u>shapeRequiredBetweenSpacing</u>	
<u>viaRequiredWithin</u>	
<u>viaSpacing</u>	allCuts, cutSizeRanges, exceptOppositeCornerNeighbors, twoCuts

---

**Virtuoso Technology Data Constraint Reference**  
Constraint Groups and Constraints

## Supported Constraints and Parameters: ICADV12.3 Only

Constraint Name	Parameters
<u>allowedBetweenNeighborWidthRanges</u>	
<u>allowedSnapPatternDefs</u>	
<u>allowedWireTypes</u>	
<u>endOfLineKeepout</u>	mask1, mask2, mask3, diffMask, forwardGap, otherEndEol, twoSides
<u>forbiddenCutClassSpacingRange</u>	longEdge, sameMask, allCuts, layer, layerWidth, mask1, mask2, mask3, paraLength, within
<u>forbiddenEdgePitchRange</u>	minWidth, outerSameMask, outerWithin, outerWidth
<u>layerMaskShiftAllowed</u>	
<u>minAdjacentFourViaSpacing</u>	
<u>minArea</u>	insideLayers, outsideLayers
<u>minClusterSpacing (One layer)</u>	clusterToCluster, edgeToEdge, sameMask
<u>minConcaveCornerSpacing</u>	
<u>minCornerSpacing (One layer)</u>	sameMask, useEdgeLength, exceptNotchLength
<u>minCutClassSpacing (One layer)</u>	horizontalOverlap, verticalOverlap, minEnclosure
<u>minEdgeAdjacentLength</u>	threeConcaveCorners, width
<u>minEndOfLineEdgeExtension</u>	minWidth, shortEdgeOnly
<u>minEndOfLineExtensionSpacing</u>	nonEol, otherWidth
<u>minEndOfLineSpacing</u>	exceptExactEolWidth, fillConcaveCorner, parallelSameMask, insideLayers, outsideLayers
<u>minExtensionEdge</u>	edgeExtension, layer
<u>minFiveWiresEndOfLineSpacing</u>	

## Virtuoso Technology Data Constraint Reference

### Constraint Groups and Constraints

Constraint Name	Parameters
<u>minNeighborExtension</u>	mask1, mask2, mask3, mask4
<u>minNotchSpacing</u>	excludeSpacing
<u>minNumCut</u>	fullyEnclosed
<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, withinFirstWidth, directionalExtension, parallelWithinTable, trimLayer, trimLengthTable, insideLayers, outsideLayers
<u>minRectArea</u>	mask1, mask2, mask3, layer, overlapType, insideLayers, outsideLayers
<u>minSideSpacing (One layer)</u>	cornerEuclidian, lengthRanges, otherLengthRanges, prlRange, sameMask
<u>minSideSpacing (Two layers)</u>	cornerEuclidian, exceptOverlap, mask1, mask2, mask3, otherMask1, otherMask2, otherMask3, insideLayers, outsideLayers
<u>minSpacing (One layer)</u>	fillConcaveCorner, mask1, mask2, mask3, ignoreIntermediateShapes

## Virtuoso Technology Data Constraint Reference

### Constraint Groups and Constraints

---

Constraint Name	Parameters
<u>minSpanLengthSpacing</u>	exactSelfSpacingTable, minSpanForExceptEoLWidth, minSpanSpacingRangesPrl, sameMask, spacingToMinSpanTable, useEdgeLength
<u>minViaSpacing (One layer)</u>	above, below, bothAboveBelow, diffMask, sizeByTouchedCorners
<u>minViaSpacing (Two layers)</u>	exceptSameNet, exceptSameMetal
<u>minVoltageExtension</u>	
<u>minVoltageSpacing (Two layers)</u>	horizontal, vertical
<u>multiMaskCheck</u>	
<u>oneSideSpacing</u>	
<u>orthogonalSnappingLayer</u>	
<u>orthogonalWSPGrid</u>	
<u>rectangularGapMinSpacing</u>	
<u>sameMaskOnLayer</u>	
<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>viaKeepoutZone</u>	
<u>viaSpacing</u>	minDistance, sameMask, sameNeighborCuts, exceptWithin

---

## **Virtuoso Technology Data Constraint Reference**

### Constraint Groups and Constraints

---

---

## Antenna Constraints

---

This chapter includes the following constraints:

- [antennaModels](#)
- [maxFloatingArea](#)

## 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.

```
antenna_ratio = (metal_area - diff_minus_factor * diff_area) / gate_area  
antenna_ratio = (cut_area - diff_minus_factor * diff_area) / gate_area
```

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.

```
antenna_ratio = metal_area / (gate_area + diff_plus_factor * 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

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 Constraint Reference

### Antenna Constraints

---

#### Parameters

<i>t_name</i>	The antenna oxide model name. Valid values are as follows: <ul style="list-style-type: none"><li>■ default: Antenna oxide1 model</li><li>■ second: Antenna oxide2 model</li><li>■ third: Antenna oxide3 model</li><li>■ fourth: Antenna oxide4 model</li></ul>
<i>t_constraintName</i>	The name of the constraint being defined. Valid values are as follows: <ul style="list-style-type: none"><li>■ antenna specifies the maximum antenna ratios for one or more antenna models for the specified layer.</li><li>■ cumulativeMetalAntenna 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.</li><li>■ cumulativeViaAntenna 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.</li><li>■ cumPerLayerAntenna specifies the maximum antenna ratios for one or more antenna models for area values accumulated for the specified layer.</li></ul>
<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)

*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 Constraint 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 "vial" 0.64 'side )
  ( diffAreaRatio "vial" ((30 0.3888) (25 0.6774)) 'side )
) ;cumPerLayerAntenna
```

## maxFloatingArea

```

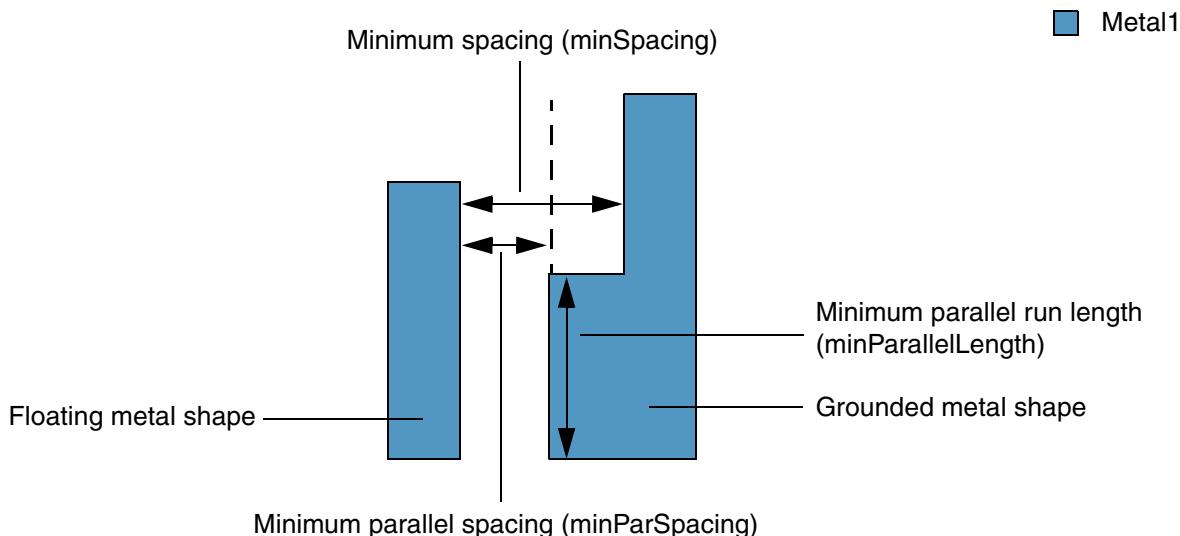
spacingTables(
    ( maxFloatingArea
        'routingLayers (tx_layer1 tx_layer2 ... tx_layerN)
        'singleLayer | 'connected | 'allConnected
        'spacings
        (
            (( "layer" ))
            (g_table)
        )
        f_area
    )
)
) ; spacingTables

```

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.



## 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

```
'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.

## 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 Constraint Reference

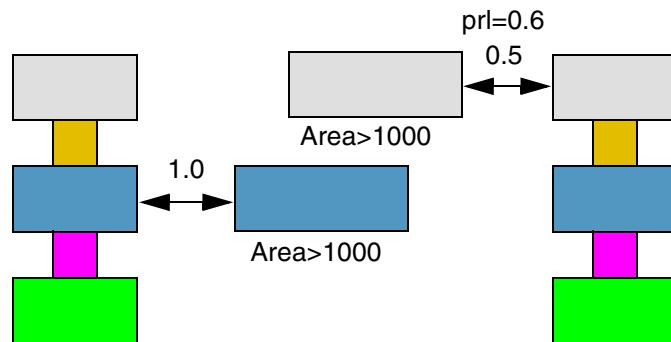
## Antenna Constraints

```

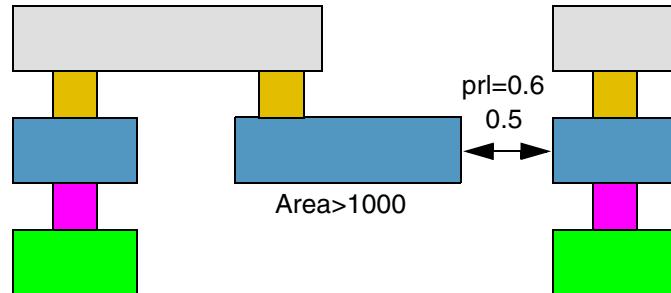
spacingTables(
    maxFloatingArea
    'routingLayers ( "Metal1" "Metal2" "Metal3" "Metal4" "Metal5" )
    'connected
    'spacings ( ((layer"))
        ("Metal1" ((1.0 0.0) (0.5 0.8) (0.2 2.0)))
        ("Metal2" ((0.6 0.0) (0.3 0.8)))
        ("Metal3" ((0.6 0.0) (0.3 0.8)))
        ("Metal4" ((0.6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

```

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).

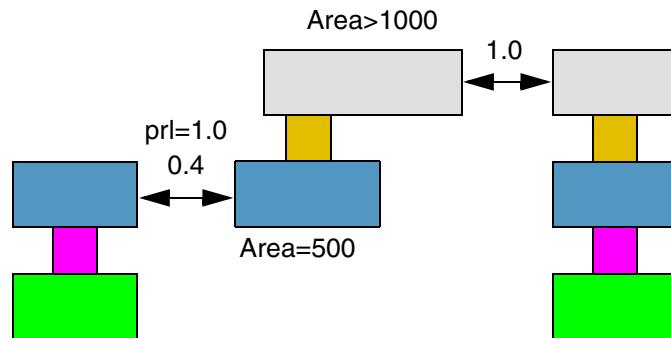


b) FAIL. When Metal1 is fabricated the large Metal1 shape is floating and is 0.5 away from the grounded Metal1 shape. This requires a prl of 0.08, but the actual prl is only 0.6.

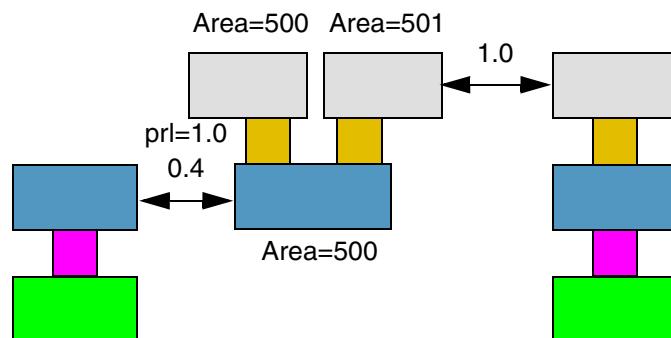
## Virtuoso Technology Data Constraint Reference

### Antenna Constraints

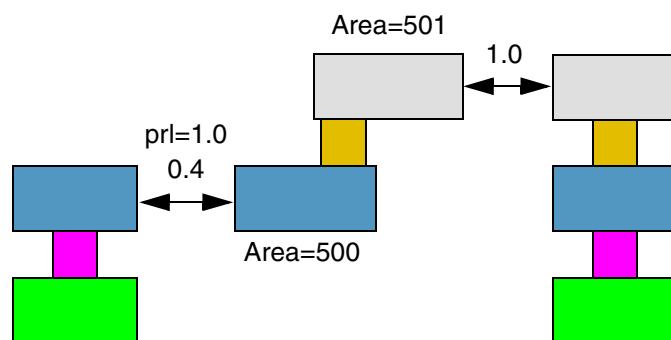
---



c) FAIL. The floating Metal2 shape satisfies the *minSpacing* requirement, but is connected to a floating Metal1 shape that does not 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 not 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.

## **Virtuoso Technology Data Constraint Reference**

### Antenna Constraints

---

---

## Area Constraints

---

This chapter includes the following constraints:

- [minArea](#)
- [minAreaEdgeLength](#)
- [minHoleArea](#)
- [minRectArea](#)

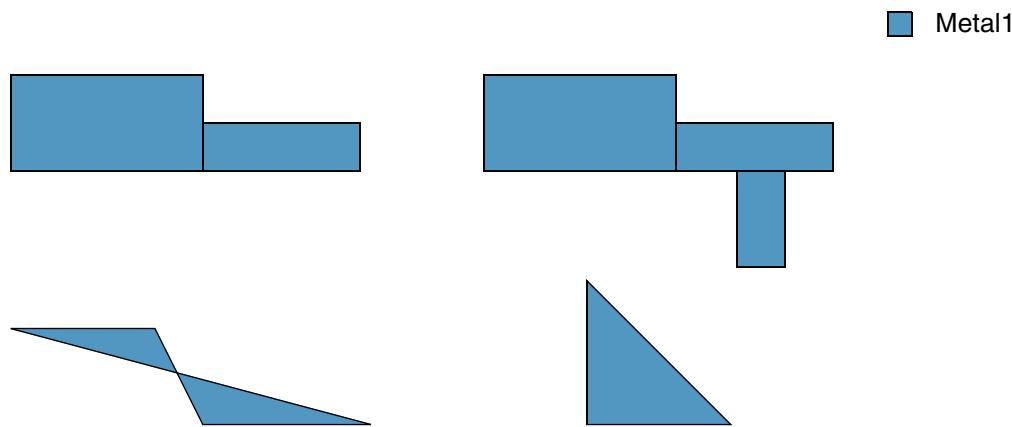
## minArea

```
spacings(
  ( minArea tx_layer
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
     | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ]
    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.

## 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.

## Parameters

```
'insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
| 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    (ICADV12.3 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 (*tx\_layer1 tx\_layer2 ... tx\_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 (*tx\_layer1 tx\_layer2 ... tx\_layerN*).

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

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

## 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.

## 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*)

(Advanced Nodes Only) 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 Constraint Reference

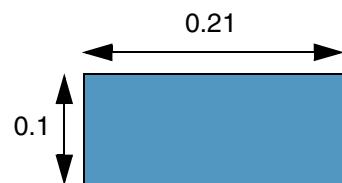
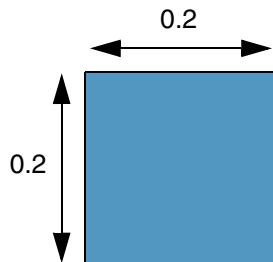
### Area Constraints

#### **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.

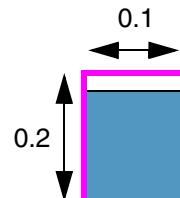
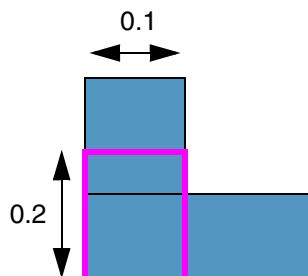
b) The constraint does not apply because one edge is 0.21 long.

#### **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 Constraint Reference

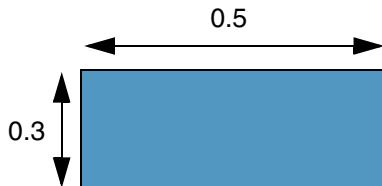
### Area Constraints

#### **Example 3: minAreaEdgeLength with exceptMinWidth**

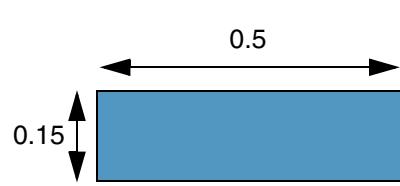
The area of a polygon must be at least 0.18 if the width of the shape is less than 0.2.

```
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).

## Virtuoso Technology Data Constraint Reference

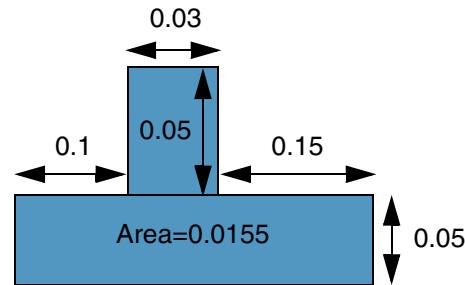
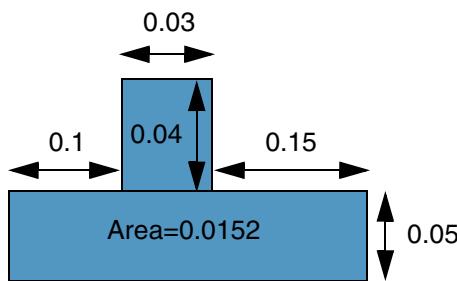
### Area Constraints

#### 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.

## Virtuoso Technology Data Constraint Reference

### Area Constraints

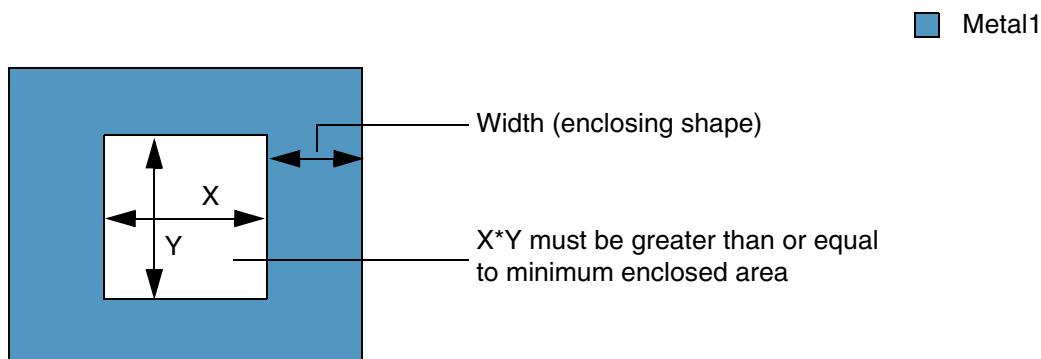
#### minHoleArea

```
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 Constraint 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 less 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 f_value</i> ) where, <ul style="list-style-type: none"><li>■ <i>f_width</i> is the width of the donut shape.</li><li>■ <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.</li></ul> Type: A 1-D table specifying floating point width and area values.

#### Parameters

<i>f_default</i>	The area 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
```

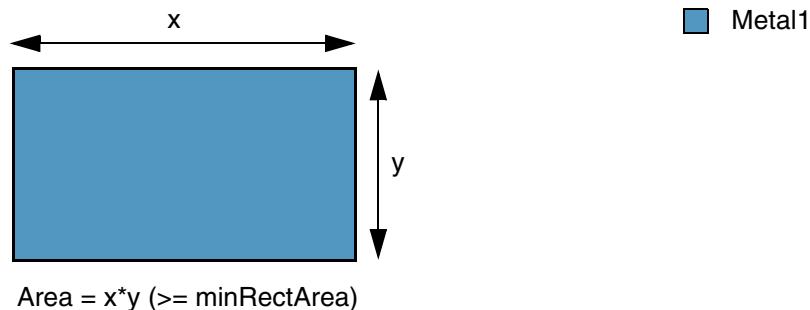
## minRectArea

```
spacings(
  ( minRectArea tx_layer
    ['mask1 | 'mask2 | 'mask3]
    ['maxWidth f_width]
    ['layer tx_overlapLayer 'overlapType x_overlapType]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ]
    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.



$$\text{Area} = x * y \ (\geq \text{minRectArea})$$

## Values

`tx_layer`

The layer on which the constraint is applied.

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

`f_area`

The minimum area that a rectangular shape on the layer must have.

## Virtuoso Technology Data Constraint Reference

### Area Constraints

---

#### Parameters

'mask1 | 'mask2 | 'mask3

(ICADV12.3 Only) The constraint applies only to the shapes on the specified mask.

Type: Boolean

'maxWidth *f\_width*

(Advanced Nodes Only) The constraint applies only to shapes with width (shorter edge) less than or equal to this value.

'layer *tx\_overlapLayer*

(ICADV12.3 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 '*overlapType*.

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

'overlapType *x\_overlapType*

(ICADV12.3 Only) The constraint applies only if the number of overlaps is equal to this value. Valid values are 1 and 2.

'insideLayers (*tx\_layer1 tx\_layer2 ... tx\_layerN*)

| 'outsideLayers (*tx\_layer1 tx\_layer2 ... tx\_layerN*)

(ICADV12.3 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 (*tx\_layer1 tx\_layer2 ... tx\_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 (*tx\_layer1 tx\_layer2 ... tx\_layerN*).

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

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

## Virtuoso Technology Data Constraint 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 areaval1.

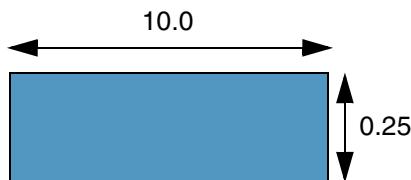
```
spacings(
  ( minRectArea "metal1" 5.0 )
  ( minRectArea "metal2" techParam("areaval1") )
) ;spacings
```

#### ***Example 2: minRectArea with maxWidth***

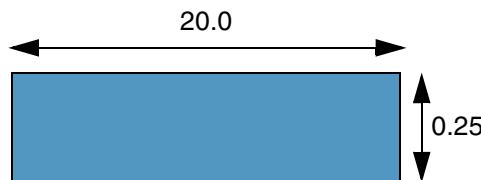
The minimum area for a rectangular shape with width less than or equal to 0.3 is 5.0.

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

 Metal1



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



b) PASS. Width is 0.25 (<0.3) and the area ( $20.0 \times 0.25$ ) is equal to the minimum specified value.



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

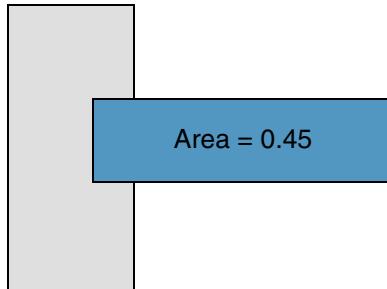
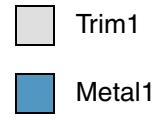
## Virtuoso Technology Data Constraint Reference

### 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 chapter includes the following constraints:

- maxDensity
- maxDiffDensity
- minDensity

## maxDensity

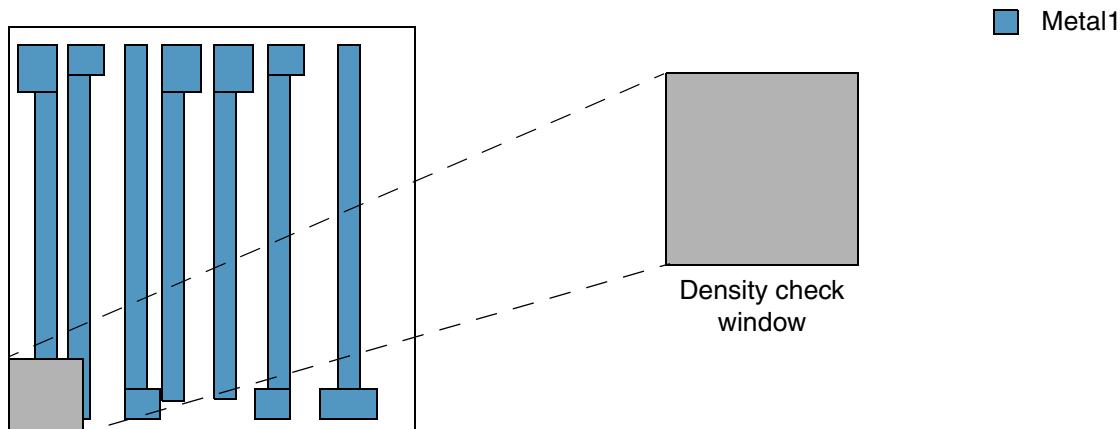
```
spacings(
  ( maxDensity tx_layer
    f_density
  )
)
) ;spacings

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 generally represented in two ways: the density for an area, which encompasses the entire design, or a window-based density, where the area under consideration is a window that is moved incrementally across the design to prevent 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 Constraint 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.

"step" nil nil ["window" nil nil]

This identifies the index for `table`.

If the `window` 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 `window` 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  $(window)^2$ .

`g_table`

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 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.

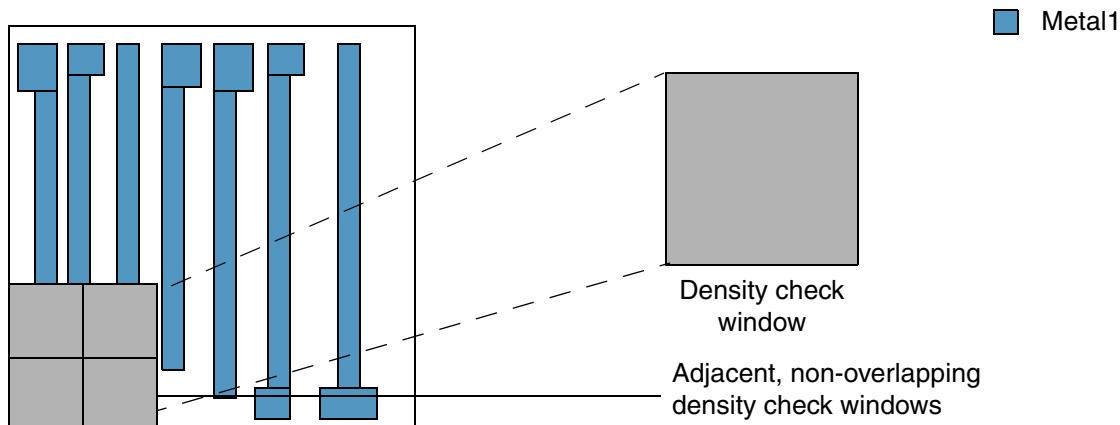
```
spacingTables(  
  ( maxDensity "Metal2"  
    (( "step" nil nil "window" nil nil )  
     65.0  
    )  
    (  
      (60.0 100.0) 65.0  
      (120.0 200.0) 130.0  
    )  
  )  
) ;spacingTables
```

## 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 Constraint Reference

### Density Constraints

---

#### Values

*tx\_layer*      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 *table*.

If the *window* 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 *window* 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  $(window)^2$ .

*g\_table*      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

*f\_default*      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
```

## minDensity

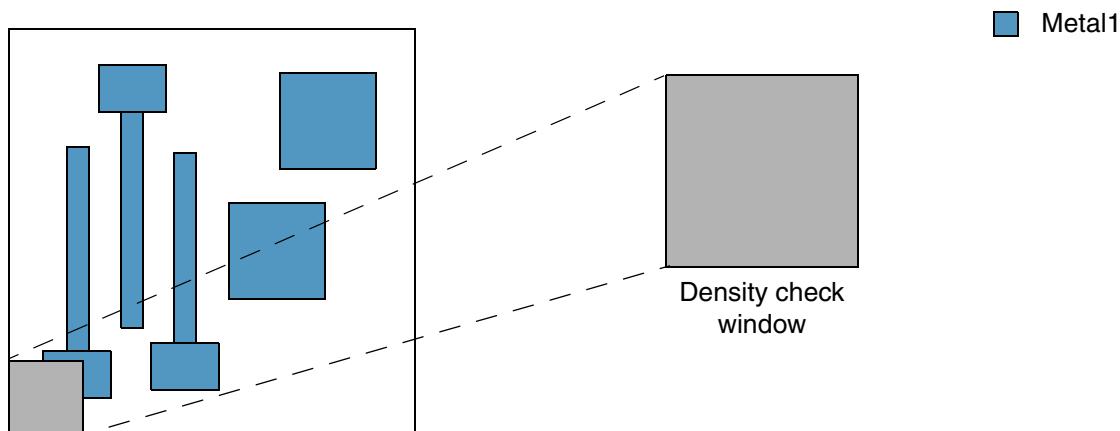
```
spacings(
  ( minDensity tx_layer
    f_density
  )
)
) ;spacings

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:

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

Density constraints are generally represented in two ways: the density for an area, which encompasses the entire design, or a window-based density, where the area under consideration is a window that is moved incrementally across the design to prevent 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 Constraint 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.
"step" nil nil ["window" nil nil]	<p>This identifies the index for <code>table</code>.</p> <p>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 <math>(2 \times step)^2</math>.</p> <p>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 <math>(window)^2</math>.</p>
<code>g_table</code>	The table row is defined as follows:  $(f\_step \ f\_window) \ f\_density$ or $f\_step \ f\_density$ where, <ul style="list-style-type: none"><li>■ <code>f_step</code> is the step size.</li><li>■ <code>f_window</code> is the window size.</li><li>■ <code>f_density</code> is the density percentage.</li></ul> 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: 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
```

---

## Extension Constraints

---

This chapter includes the following constraints:

- [dummyExtension](#)
- [maxExtension](#)
- [minCenterlineExtension](#)
- [minCornerExtension](#)
- [minEndOfLineEdgeExtension](#)
- [minEndOfLineExtension](#)
- [minExtensionDistance](#)
- [minExtensionEdge](#)
- [minExtensionOnLongSide \(Advanced Nodes Only\)](#)
- [minExtensionToCenterLine \(Advanced Nodes Only\)](#)
- [minExtensionToCorner \(Advanced Nodes Only\)](#)
- [minGateExtension](#)
- [minInnerVertexProximityExtension \(Advanced Nodes Only\)](#)
- [minInsideCornerExtension](#)
- [minNeighborExtension \(Advanced Nodes Only\)](#)
- [minOppEndOfLineExtension](#)
- [minOppExtension](#)
- [minPRBoundaryExtension](#)
- [minProtrusionWidthWithVia \(Advanced Nodes Only\)](#)
- [minQuadrupleExtension](#)

## **Virtuoso Technology Data Constraint Reference**

### Extension Constraints

---

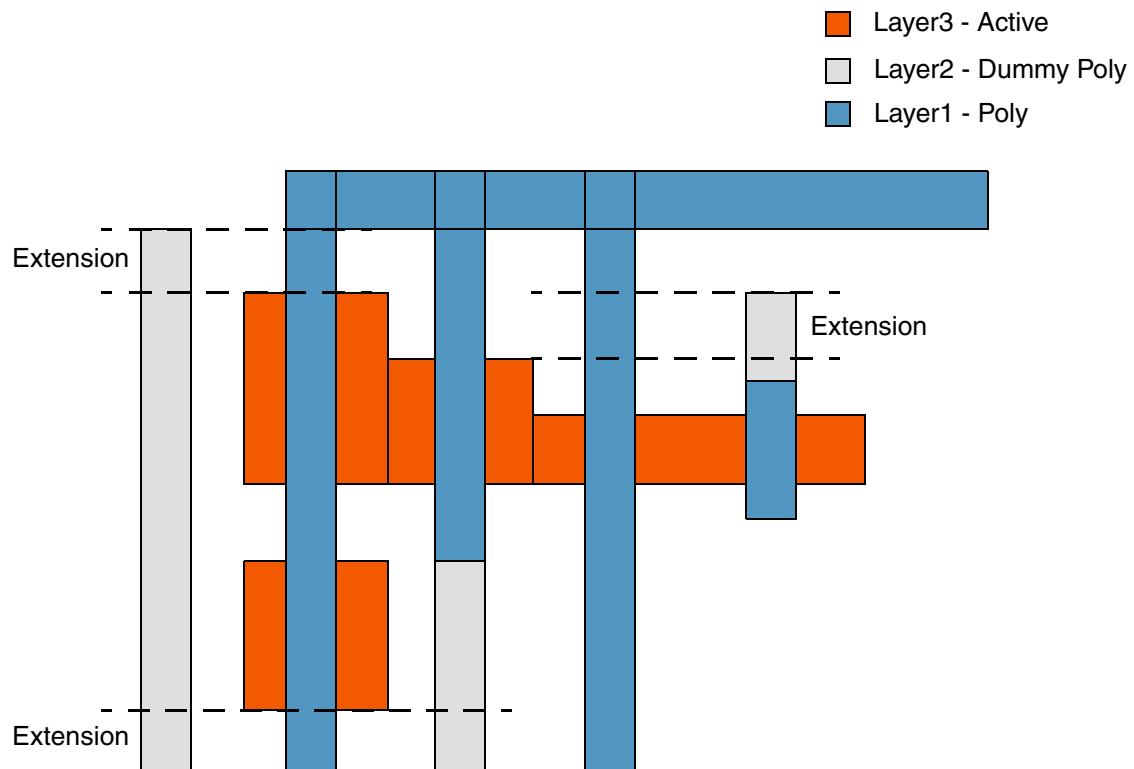
- [minSideExtension](#)
- [minTouchingDirEnclosure](#)
- [minViaExtension](#)
- [minViaJointExtension](#)
- [minVoltageExtension \(ICADV12.3 Only\)](#)
- [minWireExtension](#)

## **dummyExtension**

```
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.



## 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

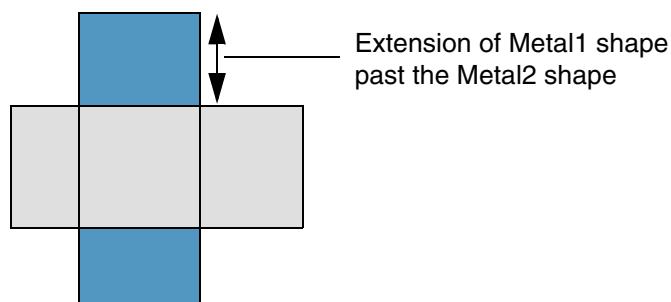
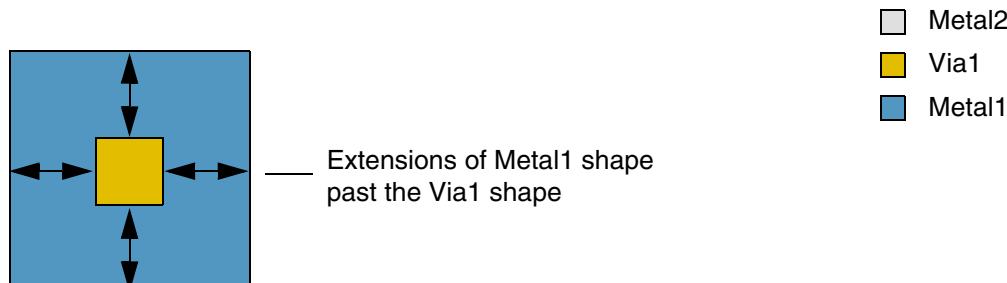
The minimum extension of a PolyX shape past a Diff shape must be at least 1.0 if the PolyX shape has width less than 0.5.

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

## 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 Constraint 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:  $(f\_width\ f\_extension)$ where, <ul style="list-style-type: none"><li>■ <i>f_width</i> is the width of the <i>layer1</i> shape. The actual width of the <i>layer1</i> shape must be greater than this value for the corresponding extension value to apply.</li><li>■ <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.</li></ul> 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 perfectly aligned double cut vias, the center of the bounding box of the two perfectly 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

*tx\_layer1*                    The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

## 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 perfectly aligned double cut vias that are less than or equal to *distance* apart must lie on the centerline of the shape with width exactly equal to *exactWidth*.

## Examples

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

## Virtuoso Technology Data Constraint Reference

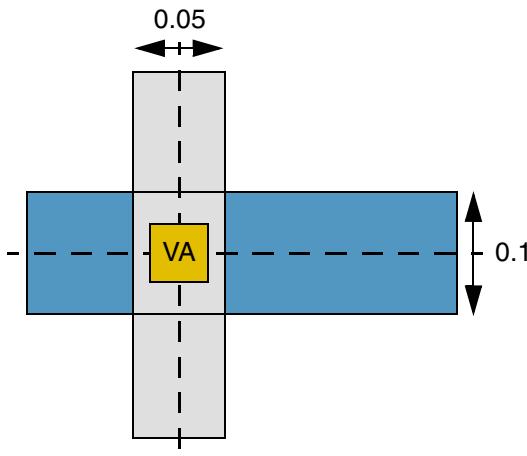
### Extension Constraints

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

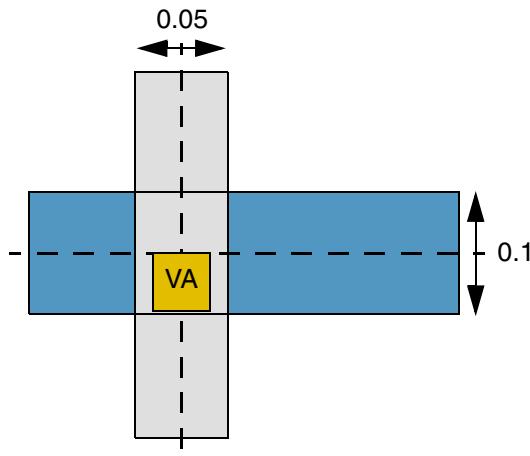
The center of a VA via cut must lie on the centerline of Metal1 and Metal2 shapes if the width of the shapes is less than or equal to 0.13.

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

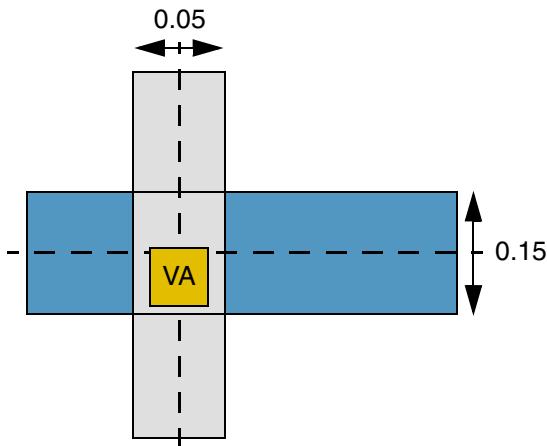
	<span style="background-color: #e0e0e0; border: 1px solid black; display: inline-block; width: 10px; height: 10px; vertical-align: middle;"></span> Metal2
	<span style="background-color: #cc9900; border: 1px solid black; display: inline-block; width: 10px; height: 10px; vertical-align: middle;"></span> Via1
	<span style="background-color: #336699; border: 1px solid black; display: inline-block; width: 10px; height: 10px; vertical-align: middle;"></span> Metal1



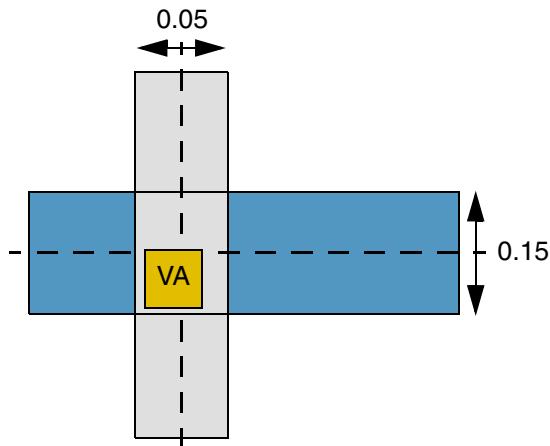
a) PASS. Both Metal1 and Metal2 shapes are less than 0.13 in width, and the center of the via cut lies on the centerlines of both shapes.



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 Constraint Reference

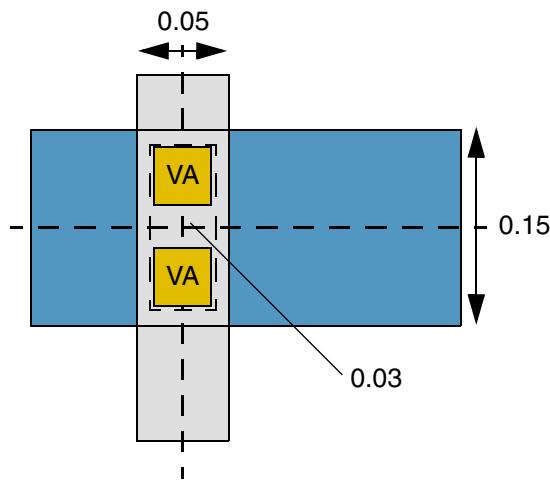
### Extension Constraints

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

The center of the bounding box of perfectly 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 exactly 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 exactly 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 Constraint Reference

### Extension Constraints

```
'corners x_corners
```

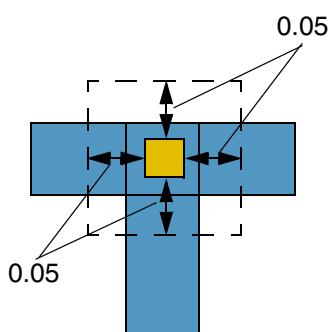
A violation occurs if the number of metal concave corners inside the search window exceeds this value.

#### Example

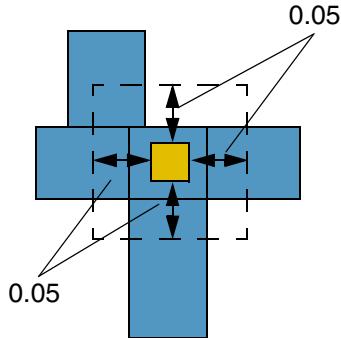
A violation occurs if more than two concave corners are found inside a search window whose sides extend 0.05 away from the edges of a VA via cut.

```
orderedSpacings(  
  ( minCornerExtension "Metall1" "Vial"  
    '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.

## minEndOfLineEdgeExtension

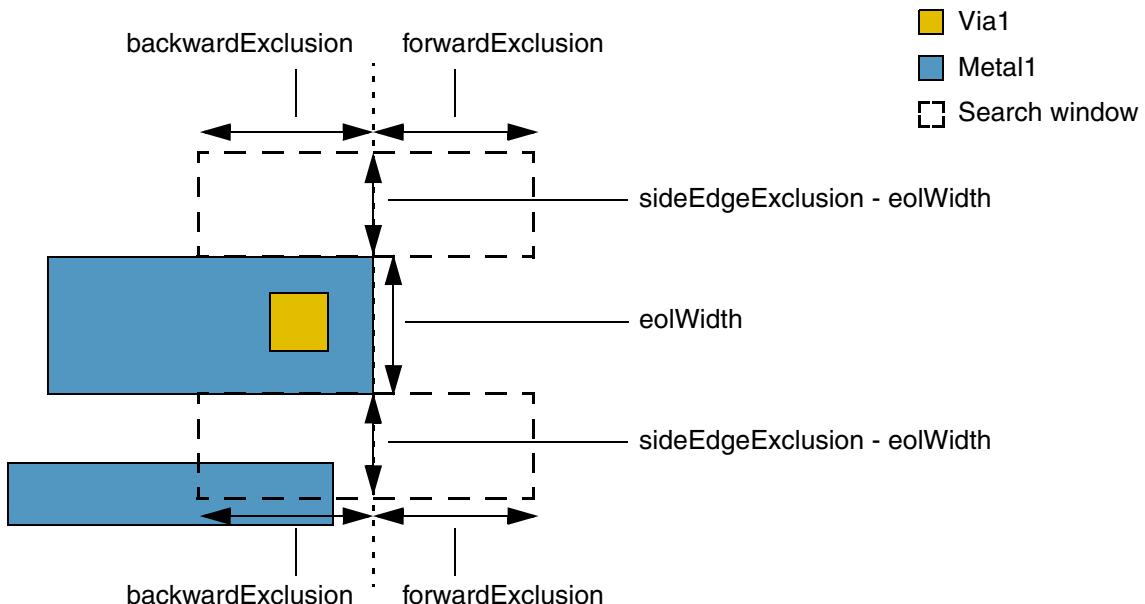
```

orderedSpacings(
    ( minEndOfLineEdgeExtension tx_layer1 tx_layer2
        'cutClass {f_width | (f_width f_length) | t_name}
        'width f_eolWidth ['minWidth f_minWidth]
        ['equalRectWidth]
        ['exactExtension f_exactExtension | 'longEdgeOnly | 'shortEdgeOnly]
        ['sideExclusion (f_sideEdgeExclusion f_backwardExclusion
                        f_forwardExclusion)
    ]
    ['length f_length]
    ['maxRectEnclosureArea f_area]
    ['concaveCornerLength f_concaveCornerLength]
    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 Constraint 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*

(ICADV12.3 Only) The constraint applies only if the actual 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.

'equalRectWidth

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

'exactExtension *f\_exactExtension*

The extension must be exactly 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.

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

'shortEdgeOnly                    (ICADV12.3 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.

'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.

## 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, and minWidth](#)

## Virtuoso Technology Data Constraint Reference

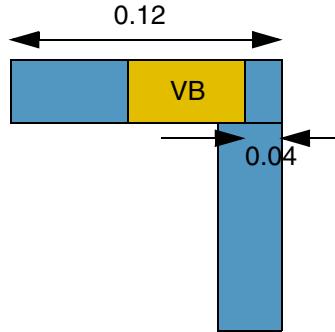
### Extension Constraints

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

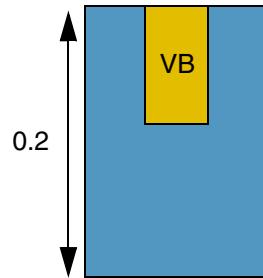
The long edge of a rectangular cut of type VB must have an extension of at least 0.04 on an end-of-line edge less than 0.15 wide.

```
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 Constraint 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 actual 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
```

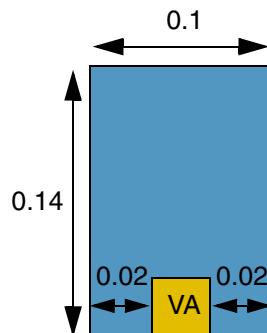
 Via1  
 Metal1



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 Constraint Reference

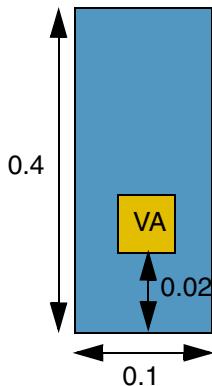
### Extension Constraints

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

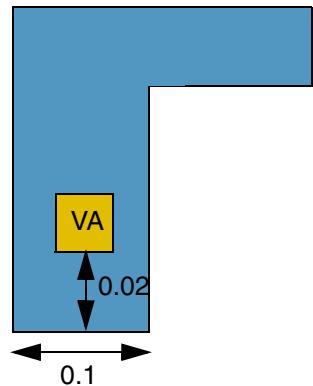
A cut of type VA must have an extension of at least 0.025 if the end-of-line edge is less than 0.2 wide and the enclosing Metal1 shape is a rectangle with area less than 0.05.

```
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 (<0.25).



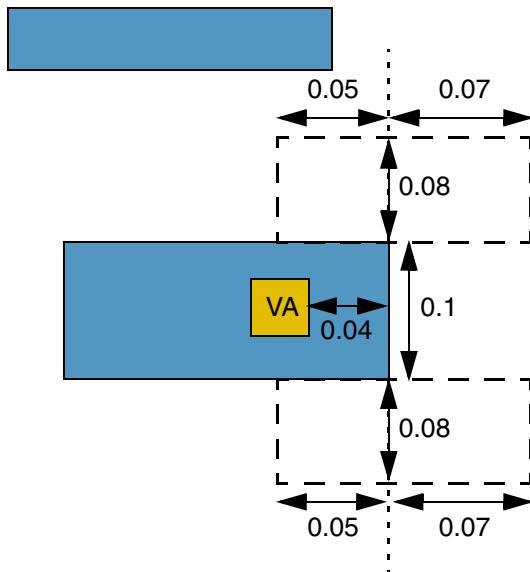
b) The constraint does not apply because the enclosing shape is not rectangular.

**Example 4: *minEndOfLineEdgeExtension* with *cutClass*, *width*, and *sideExclusion***

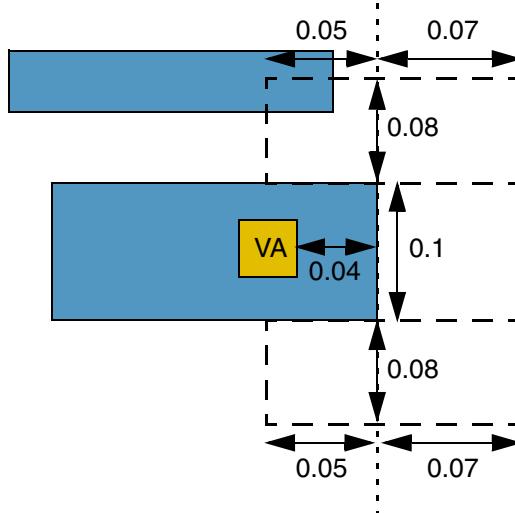
A cut of type VA must have an extension of at least 0.05 if the end-of-line edge is less than 0.15 wide and a neighboring parallel shape is present inside a search window with dimensions 0.18 (side edge exclusion), 0.05 (backward exclusion), and 0.07 (forward exclusion).

```
orderedSpacings (
  ( minEndOfLineEdgeExtension "Vial" "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).

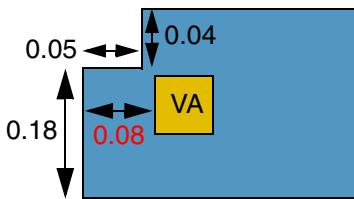
**Example 5: *minEndOfLineEdgeExtension* with *cutClass*, *width*, and *concaveCornerLength***

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:

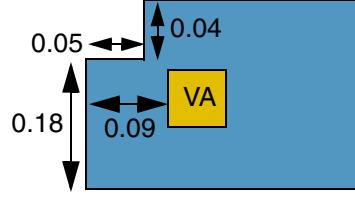
- The end-of-line edge is less than 0.2 wide.
- The edge adjoining the end-of-line edge forms a concave corner, with both concave edges less than 0.06 in length.

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

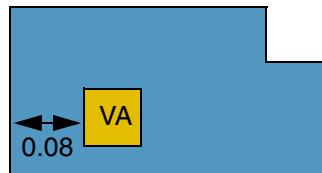




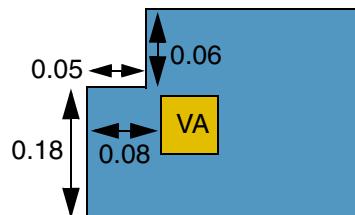
a) FAIL. The end-of-line width is 0.18 (<0.2) and the lengths of the concave sides are 0.04 and 0.05 (both less than 0.06). However, the Metal1 extension is only 0.08 (<0.09).



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 exactly equal to 0.06 (and not less than 0.06).

## Virtuoso Technology Data Constraint Reference

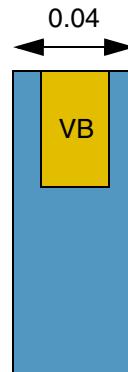
### Extension Constraints

#### ***Example 6: minEndOfLineEdgeExtension with cutClass, width, and minWidth***

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
```

 Via1  
 Metal1



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

## **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.  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.

### **Parameters**

<i>f_eolWidth</i>	The constraint applies only if the end-of-line width is less than this value.
-------------------	---

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

*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 Constraint Reference

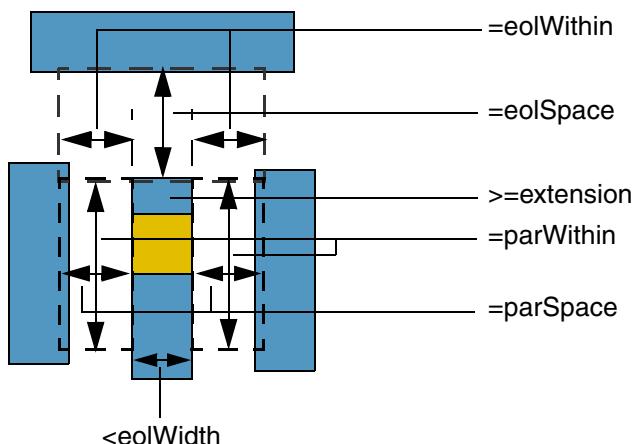
### Extension Constraints

#### 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 tx_layer1 tx_layer2
    f_eolWidth
    f_eolSpace f_eolWithin
    f_parWithin f_parSpace
    f_extension
  )
) ;orderedSpacings
```

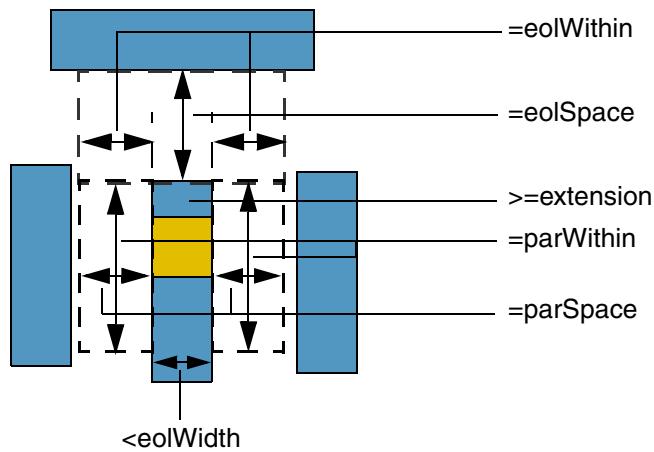
 Via1  
 Metal1  
 Verification region



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

## Virtuoso Technology Data Constraint Reference

### Extension Constraints



- 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]
    ['cutClass {f_width | (f_width f_length) | t_name}']
    ['exceptEdgeLengthRange g_exceptEdgeLengthRange]
    f_extension
    ['coincidentAllowed] ['manhattan]
  )
)

; orderedSpacings

spacingTables(
  ( minExtensionDistance tx_layer1 tx_layer2
    (( "width" nil nil ))
    ( g_table )
    ['coincidentAllowed] ['manhattan]
  )
)

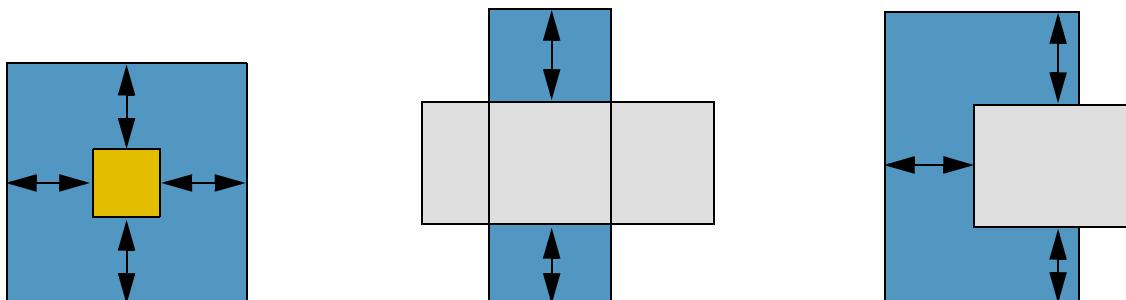
; spacingTables

```

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.

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.

- Metal2
- Via1
- Metal1



## 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> . The <i>g_table</i> row is defined as: $(f\_width\ f\_extension)$ where, <ul style="list-style-type: none"><li>■ <i>f_width</i> is the width of the <i>layer1</i> shape. The actual width of the <i>layer1</i> shape must be greater than or equal to this value for the corresponding extension value to apply.</li><li>■ <i>f_extension</i> is the extension of the <i>layer1</i> shape past the <i>layer2</i> shape. The extension must be greater than or equal to this value.</li></ul> Type: A 1-D table specifying floating-point width and extension values.

## Parameters

'horizontal | 'vertical

(Advanced Nodes Only) 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

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

'cutClass {*f\_width* | (*f\_width f\_length*) | *t\_name*}

(Advanced Nodes Only) 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*

(Advanced Nodes Only) 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.

'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 Constraint Reference

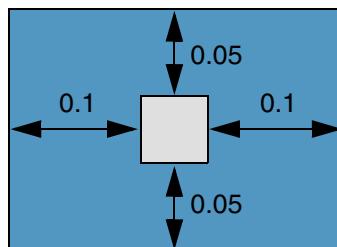
### Extension Constraints

#### 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 extensions in the horizontal direction.

## 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]
        f_extension
    )
)
;spacings
```

## Virtuoso Technology Data Constraint Reference

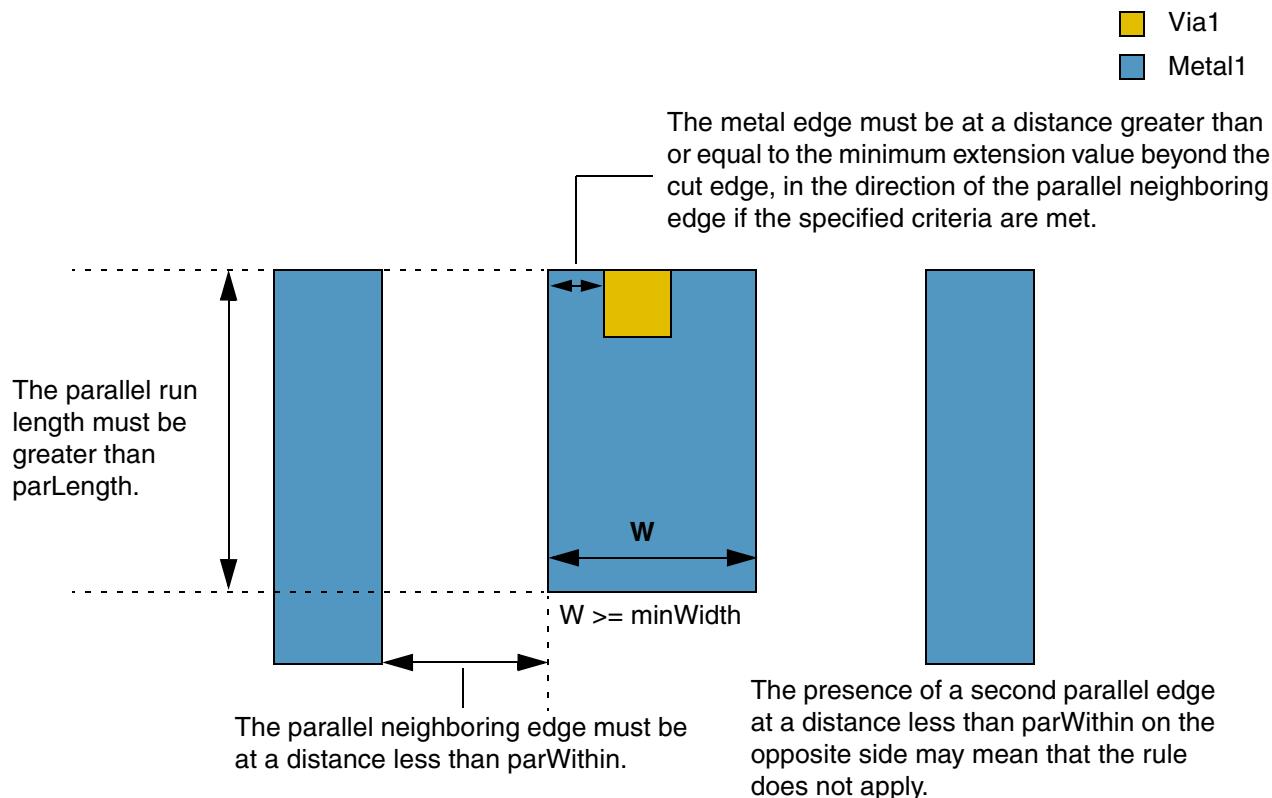
### Extension Constraints

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:

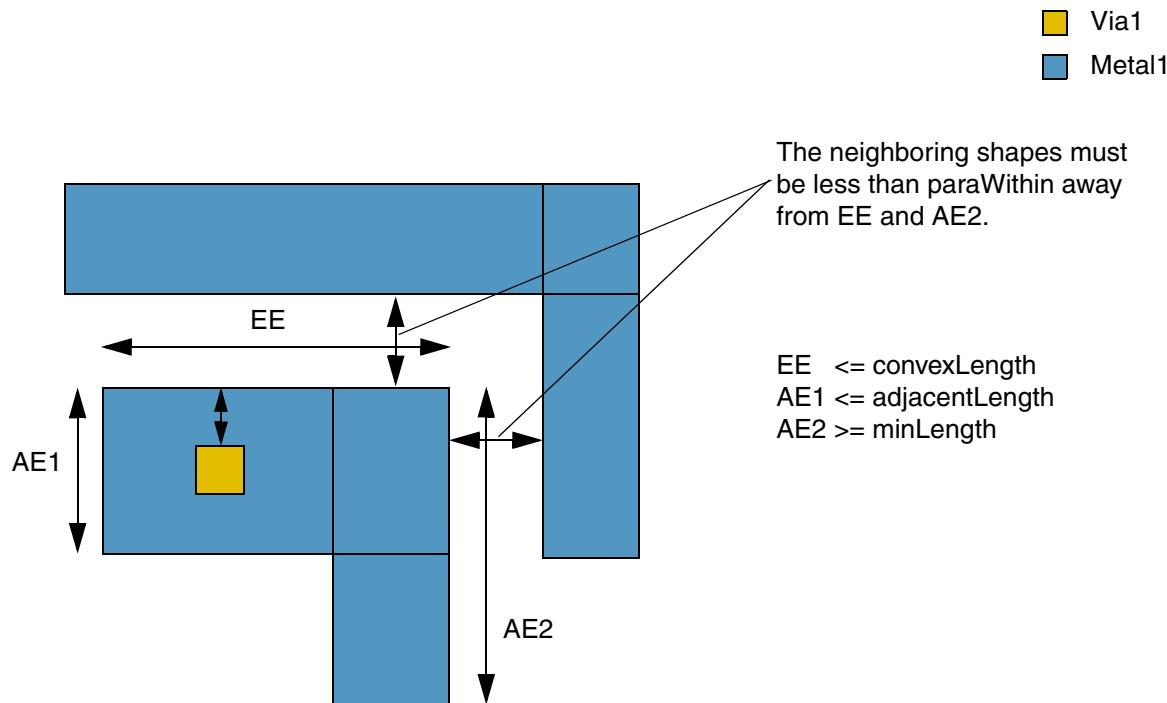
- An extra cut shape exists in the same metal extension.
- The enclosing metal shape has two neighboring shapes, on opposite sides, less than `parWithin` away and with parallel run length greater than `parLength`.
- The cut shape belongs to a class different from the one specified.



## Virtuoso Technology Data Constraint Reference

### Extension Constraints

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

*tx\_layer1* The first layer (metal) on which the constraint is applied.

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

*tx\_layer2* The second layer (cut) on which the constraint is applied.

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

*g\_extension* The minimum extension value that must be satisfied.

## Virtuoso Technology Data Constraint 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 or equal to 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\_parLength*

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

'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 Constraint 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	(Advanced Nodes Only) 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 ' <i>twoSides</i> ' extension requirement is greater than or equal to this value.

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

'exceptConcaveCorner *f\_exceptConcaveCorner*

(Advanced Nodes Only) 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*

(ICADV12.3 Only) The metal layer to which 'cutToMetalSpace spacing must be measured.

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

'edgeExtension *f\_edgeExtension*

(ICADV12.3 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.

'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

(Advanced Nodes Only) The direction in which the extension is measured.

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

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

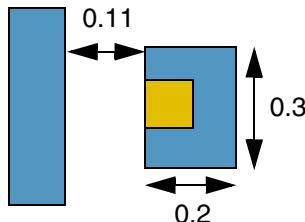
---

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

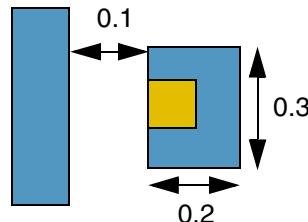
The extension of a Metal1 shape with minimum width 0.2 past a Via1 cut must be at least 0.05 in the direction of a neighboring Metal1 edge less than 0.11 away, when the parallel run length for the Metal1 shapes is greater than 0.25.

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

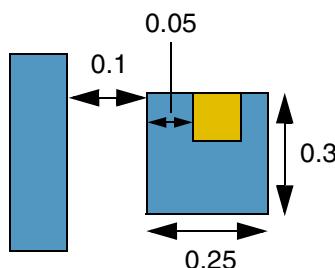
 Via1  
 Metal1



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).



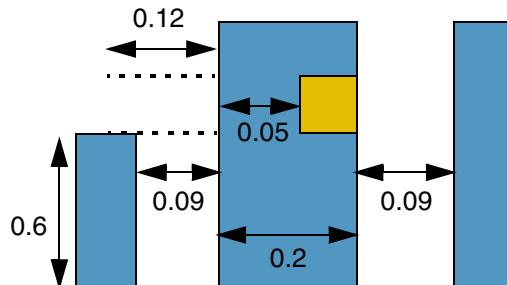
c) PASS. The extension of the metal edge, to which the constraint applies, past the cut shape is 0.05.

**Example 2: minExtensionEdge with exceptTwoEdges**

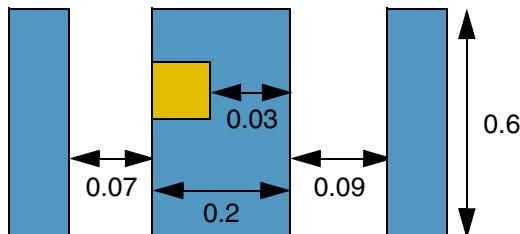
The extension of a Metal1 shape with minimum width 0.2 past a Via1 cut must be at least 0.05 in the direction of a neighboring Metal1 edge greater than or equal to 0.08 and less than 0.1 away from it, when the parallel run length for the Metal1 shapes is greater than 0.5. The constraint does not apply 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
```

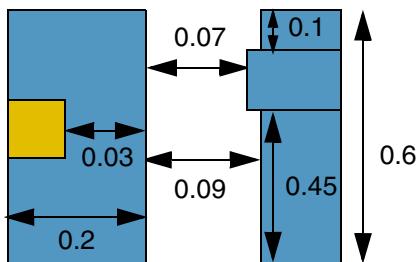
 Via1  
 Metal1



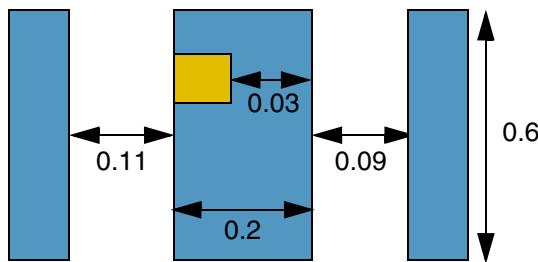
a) FAIL. The right neighbor does not have a left neighbor in the dotted region, so 'exceptTwoEdges' is not triggered. This implies that there must be a 0.05 enclosure on the right.



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.



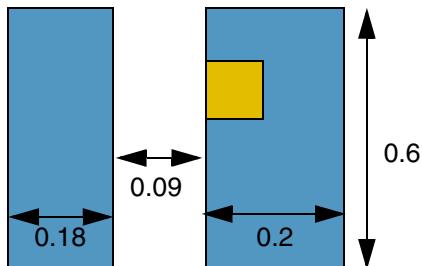
d) The constraint does not apply. 'exceptTwoEdges' is triggered and both neighbors are at a distance less than 0.12 from the enclosing shape. A violation would occur if 'exceptWithin' was not defined.

**Example 3: *minExtensionEdge* with *paraEdgeWidth* and *includeCorner***

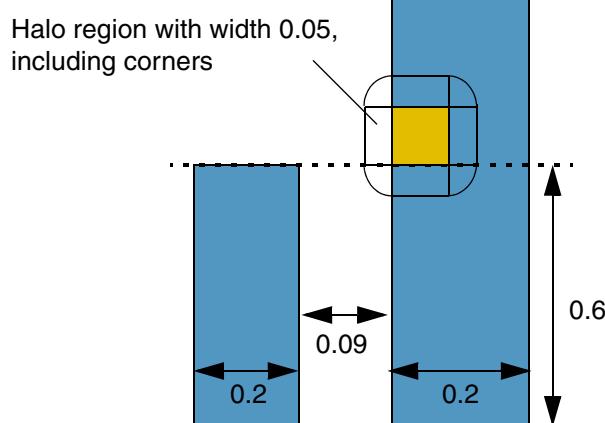
The extension of a Metal1 shape with minimum width 0.2 past a Via1 cut must be at least 0.05, for edges as well as corners, in the direction of a neighboring Metal1 edge less than 0.1 away from it, when the parallel run length for the Metal1 shapes is greater than 0.5. The width of the neighboring shape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

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

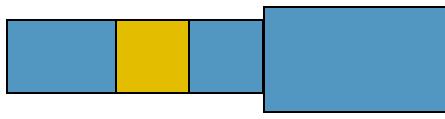
---

#### ***Example 4: minExtensionEdge with twoSides and eolWidth***

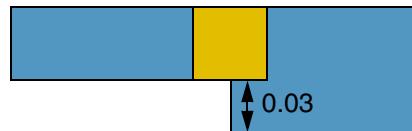
The extension of a Metal1 shape past a Via1 cut must be at least 0.02 along the entire length of a pair of opposite edges of the cut shape, and the end-of-line width of the metal edges that satisfy the 'twoSides' requirement must be at least 0.15.

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

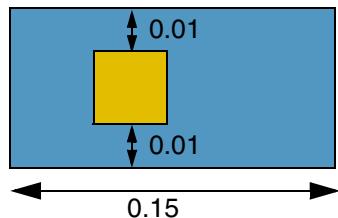
 Via1  
 Metal1



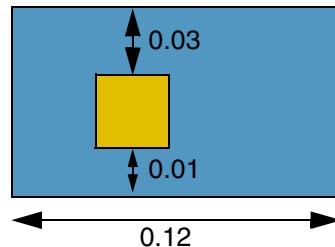
a) PASS. The metal shape has an extension of 0 along the entire length of a pair of opposite edges of the cut shape.



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.

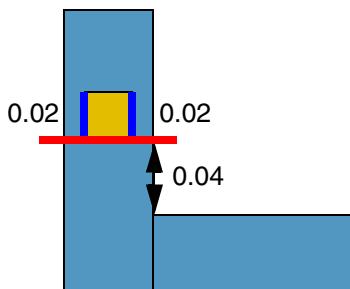
**Example 5: *minExtensionEdge* with *exceptConcaveCorner* and *cutToMetalSpace***

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:

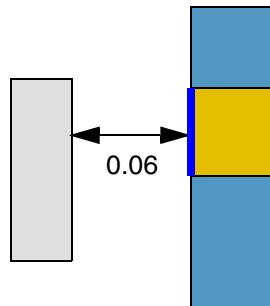
- The distance of a metal concave corner from a cut edge orthogonal to the cut edges satisfying the extension requirement must be greater than 0.05.
- The distance of a cut edge, satisfying the extension requirement, from a different-net wire must be greater than or equal to 0.07.

```
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).

## Virtuoso Technology Data Constraint Reference

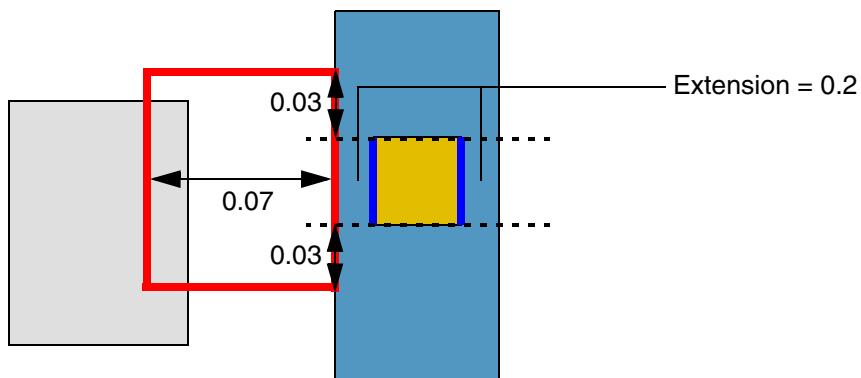
### Extension Constraints

#### **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



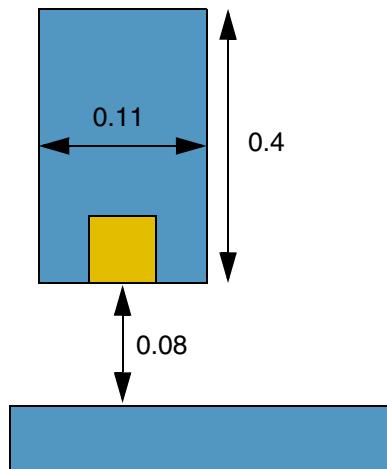
**FAIL.** The blue cut edges fulfill the 0.02 extension requirement, but the distance measured from Metal1 edge to the Metal2 object, after applying an extension of 0.03 on the Metal1 edge along the direction of the cut edge, is less than 0.07. In other words, no neighbor must lie in the region defined 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 ( $\geq 0.2$ ). However, the extension of the bottom metal edge, to which the constraint applies, past the cut shape is 0 ( $< 0.02$ ).

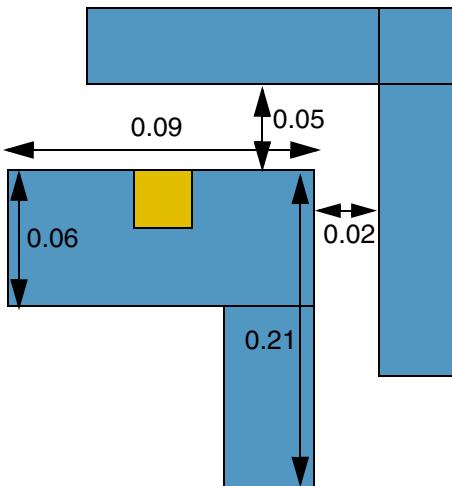
**Example 8: minExtensionEdge with convexLength**

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:

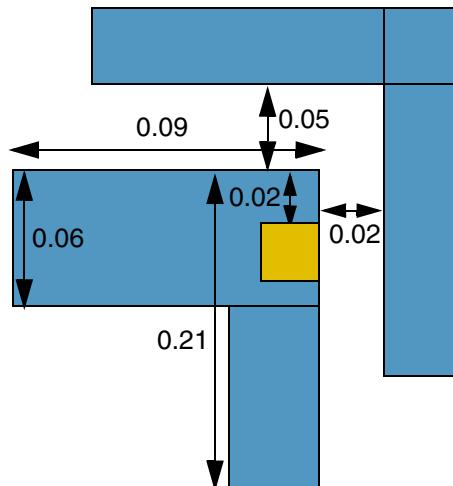
- EE is less than or equal to 0.1 and lies between two convex corners.
- EE has two adjacent edges, one of which (AE1) also lies between two convex corners and is less than or equal to 0.08. The other adjacent edge (AE2) is greater than or equal to 0.2.
- The distance of EE and AE2 from a neighboring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



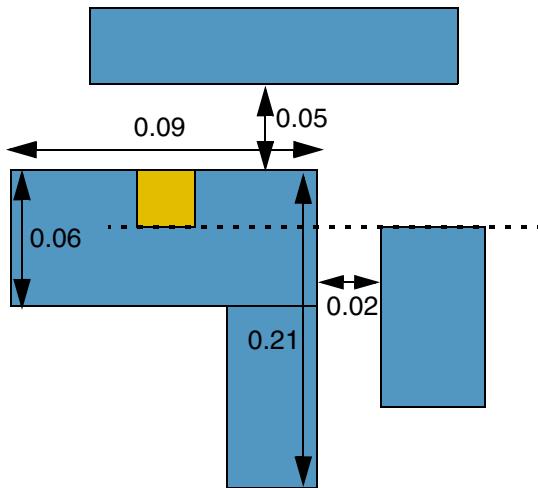
a) FAIL. All conditions are satisfied, but the extension of the enclosing edge on the top edge of the cut shape is 0.



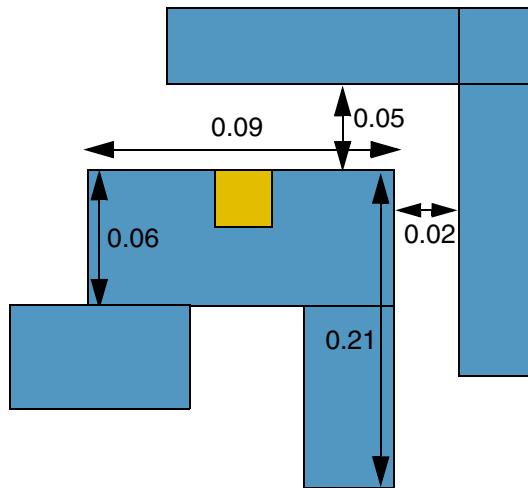
b) PASS. The extension of the enclosing edge on the top edge of the cut shape is 0.02.

## Virtuoso Technology Data Constraint 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.

## **minExtensionOnLongSide (Advanced Nodes Only)**

```
orderedSpacings(
  ( minExtensionOnLongSide tx_layer1 tx_layer2
    ['widthRanges (g_ranges) ['otherWidthRanges (g_otherWidthRanges)]]
    f_extension
  )
) ;orderedSpacings
```

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 <i>g_ranges</i>	The constraint applies only if the width of the <i>layer1</i> shape falls in one of these ranges.  Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint.
'otherWidthRanges <i>g_otherWidthRanges</i>	The constraint applies only if the width of the <i>layer2</i> shape falls in one of these ranges.  Type: Floating-point values specifying a <u>range</u> of widths that trigger the constraint.

## Examples

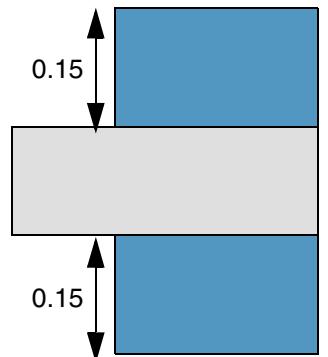
- [Example 1: minExtensionOnLongSide](#)
- [Example 2: minExtensionOnLongSide with widthRanges and otherWidthRanges](#)

### ***Example 1: minExtensionOnLongSide***

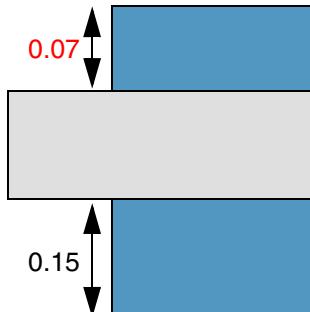
A Metal1 shape must have an extension greater than or equal to 0.15 on the long edges of a Metal2 shape.

```
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).

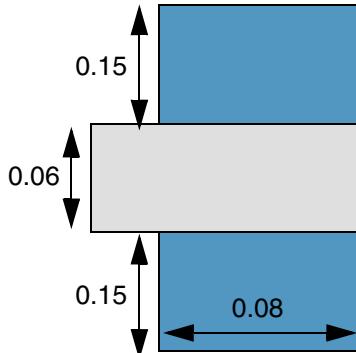
**Example 2: *minExtensionOnLongSide* with *widthRanges* and *otherWidthRanges***

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:

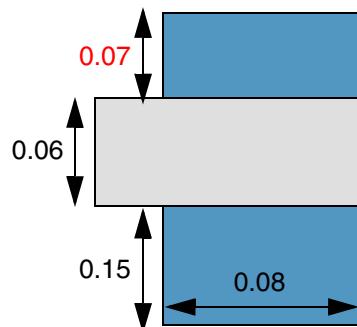
- Width of the Metal1 shape is greater than or equal to 0.08.
- Width of the Metal2 shape is either 0.04 or 0.06.

```
orderedSpacings(
    ( minExtensionOnLongSide "Metal1" "Metal2"
        'widthRanges ( ">= 0.08" )
        'otherWidthRanges ( 0.04 0.06 )
        0.15
    )
) ;orderedSpacings
```

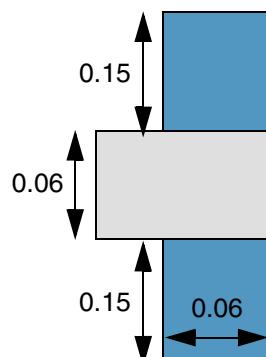
Metal2  
 Metal1



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 (Advanced Nodes Only)

```
orderedSpacings(
  ( minExtensionToCenterLine tx_layer1 tx_layer2
    ['horizontal | 'vertical]
    ['widthRanges (g_ranges)]
    ['otherWidthRanges (g_otherRanges)]
    f_extension
  )
) ; orderedSpacings
```

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

'horizontal   'vertical	The direction in which the extension is measured. If direction is not specified, the extension is measured in any direction. Type: Boolean
'widthRanges (g_ranges)	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 Constraint 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 Constraint Reference

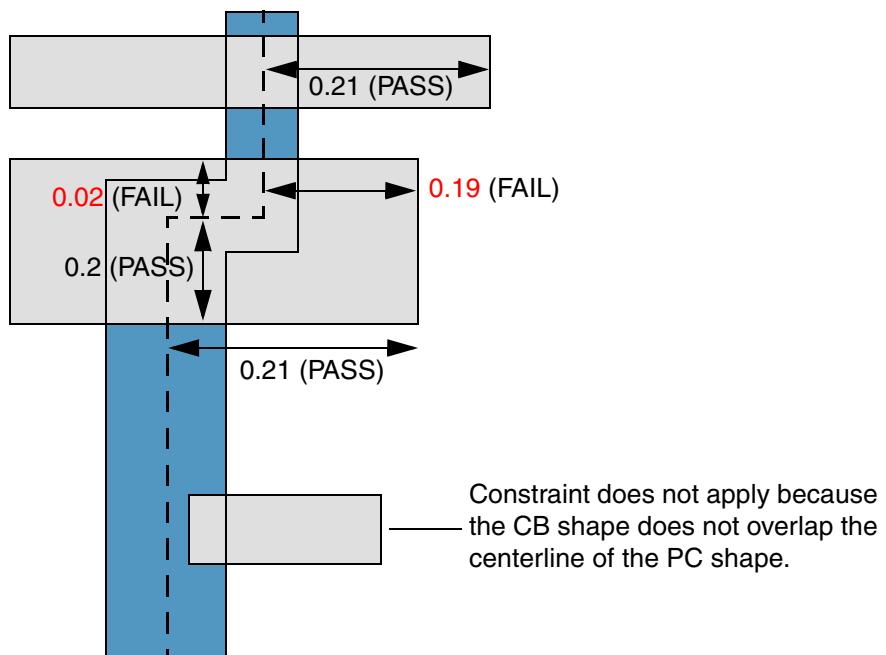
### Extension Constraints

#### ***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 Constraint Reference

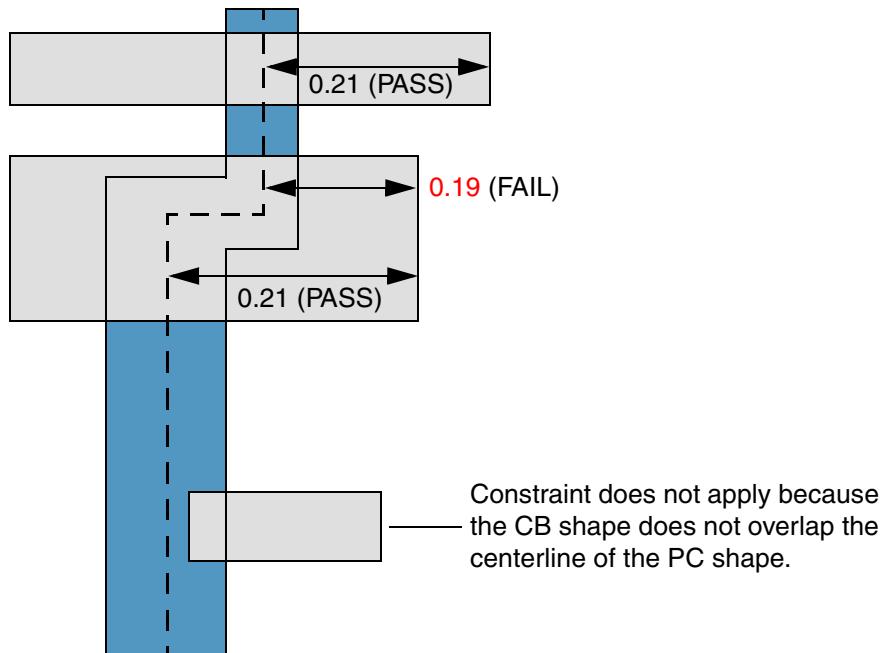
### Extension Constraints

#### ***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 Constraint Reference

### Extension Constraints

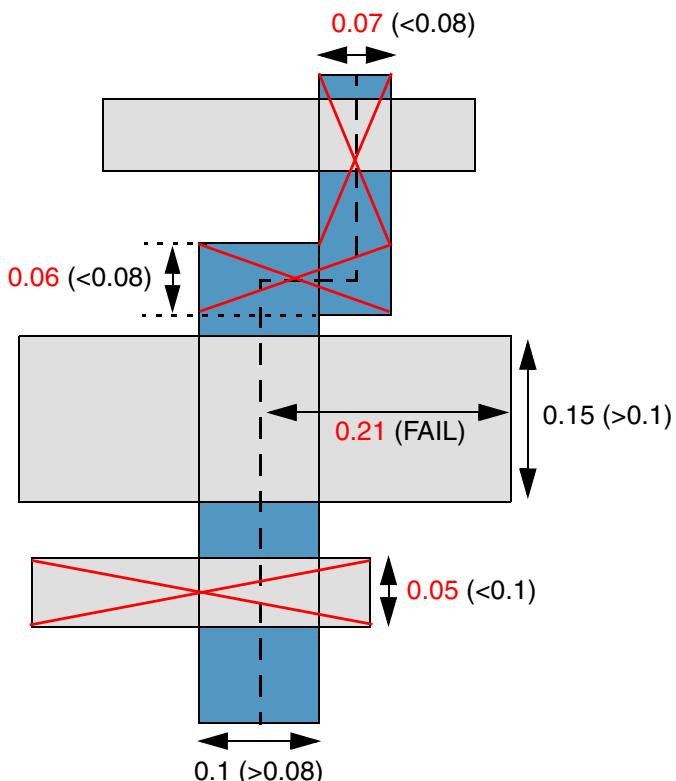
---

#### ***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 (Advanced Nodes Only)

```
orderedSpacings(
  ( minExtensionToCorner tx_layer1 tx_layer2
    { 'toConcaveCorner
      {[ 'parallelExtension f_parExt 'width f_width 'length f_length]
       | ['convexLength f_convexLength 'enclosure (f_enc11 f_enc12)]
       }
     | '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
```

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

*tx\_layer1*

The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

*f\_spacing*

The value that determines how far a *layer2* shape must be from a corner of the enclosing *layer1* shape. The type of corner, concave or convex, determines how the spacing between the *layer2* shape and the corner is measured.

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

#### Parameters

'toConcaveCorner      The distance between an edge of a *layer2* shape and a concave corner of the enclosing *layer1* shape must be greater than or equal to the constraint value.

Type: Boolean

'parallelExtension *f\_parExt*

The concave corner is extended by the constraint value on both sides along the edge of the *layer1* shape perpendicular to the edge with width *width* to form a search window; the other side of the search window is equal to the actual width of the *layer1* shape.

If a *layer2* shape is found inside the search window, then the *layer2* edge parallel to the *layer1* edge that forms the concave corner (and belongs to the *layer1* shape with width less than or equal to *width*) must have an enclosure greater than or equal to *parExt*. Enclosure is checked on a *layer2* edge only if that edge overlaps the search window.

'width *f\_width*

The constraint applies only if the width of the enclosing *layer1* shape, an edge of which forms one part of the concave corner, is less than or equal to this value.

'length *f\_length*

The constraint applies only if the length of the *layer1* edge that forms the other part of the concave corner is greater than or equal to this value.

'convexLength *f\_convexLength*

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 (*f\_enc11 f\_enc12*)

If an edge of a *layer2* shape has an enclosure less than *enc11* on an edge that forms a convex corner, the adjacent *layer2* edges must have enclosures greater than *enc12*.

## Virtuoso Technology Data Constraint 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 Constraint 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](#)

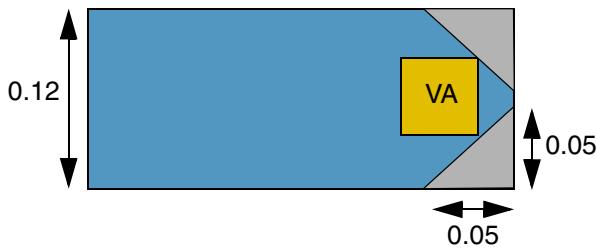
**Example 1: minExtensionToCorner with toConvexCorner, eolWidth, and minLength**

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.

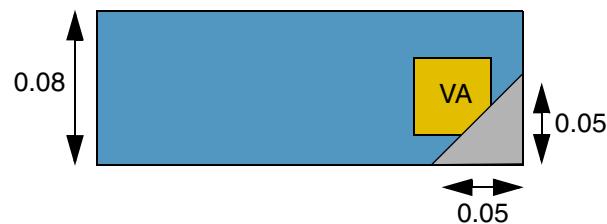
**Note:** Both 'toConvexCorner' and 'eolWidth' are specified; therefore, the keepout region is a triangle.

```
orderedSpacings(
  ( minExtensionToCorner "Metal1" "Via1"
    'toConvexCorner
    'eolWidth 0.07 'minLength 0.05
    'cutClass "VA"
    0.05
  )
) ;orderedSpacings
```

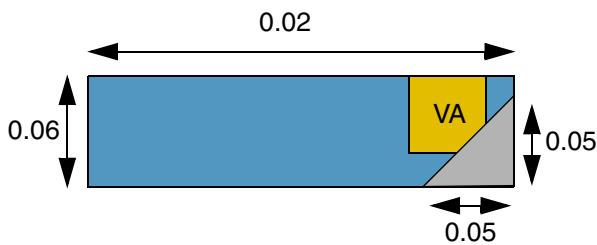
	Via1 Metal1 Keepout region
---	----------------------------------



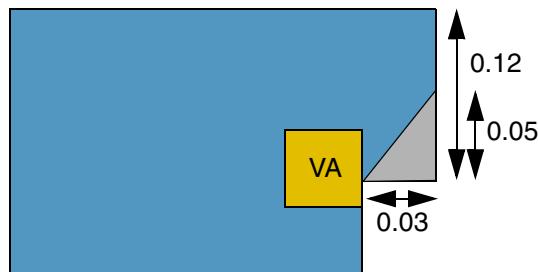
a) PASS. The via cut is outside both keepout regions.



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 Constraint Reference

### Extension Constraints

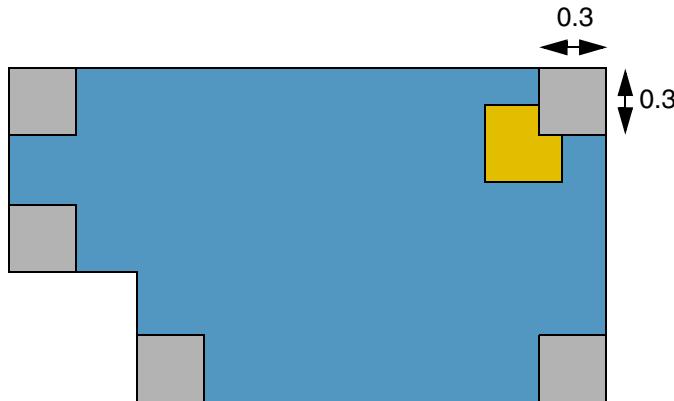
#### ***Example 2: minExtensionToCorner with toConvexCorner***

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.

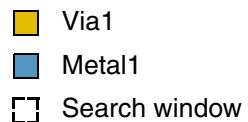
**Example 3: *minExtensionToCorner* with *toConcaveCorner*, *parallelExtension*, *width*, and *length***

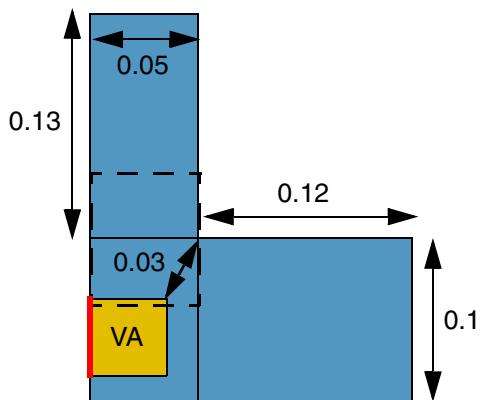
The distance of a VA via cut on layer Via1 from the concave corners of an enclosing Metal1 shape must be at least 0.03. The constraint applies only if the following conditions are true:

- 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.

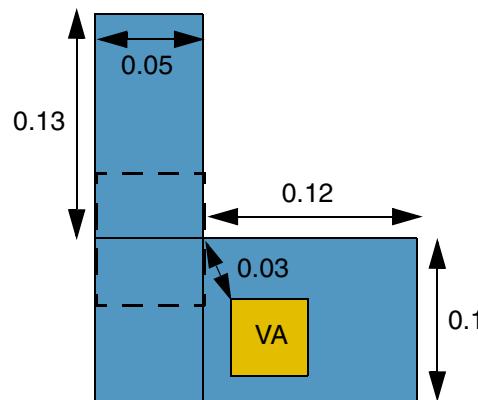
Additionally, the edge of the via cut parallel to the metal edge with width less than or equal to 0.08 has enclosure greater than or equal to 0.02.

```
orderedSpacings(
    ( minExtensionToCorner "Metal1" "Via1"
        'toConcaveCorner
        'parallelExtension 0.02 'width 0.08 'length 0.05
        'cutClass "VA"
        0.03
    )
); orderedSpacings
```





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).

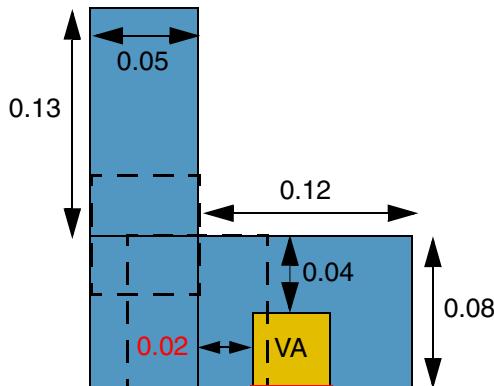


b) The constraint does not apply because the via cut lies outside the search window.

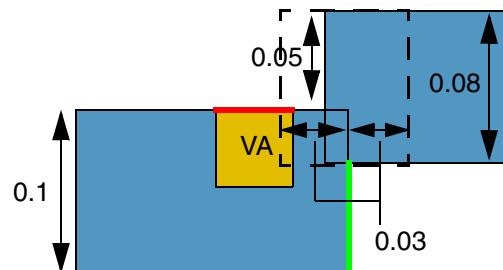
## Virtuoso Technology Data Constraint Reference

### 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 ( $<0.02$ ). (Note that 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.

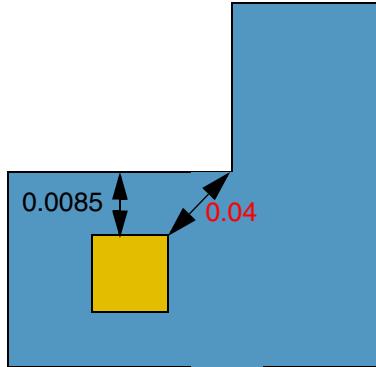
```

spacings(
  ( viaEdgeType "Via1"
    'edgeExtension 0.009
    1
  )
) ;spacings

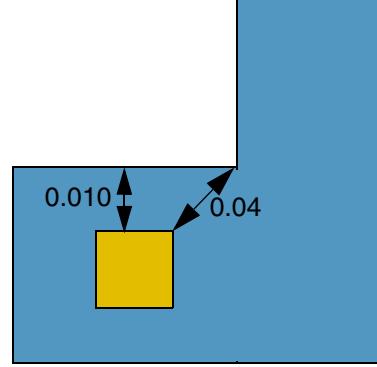
orderedSpacings(
  ( minExtensionToCorner "Metal1" "Via1"
    'toConcaveCorner
    'viaEdgeType 1
    'exceptCornerTouch
    0.05
  )
) ;orderedSpacings

```

 Via1  
 Metal1



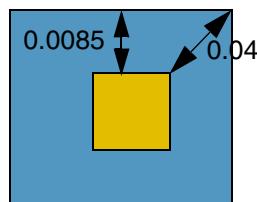
a) FAIL. The extension of the top edge is 0.0085 (<0.009). However, the distance of this edge from the concave corner is 0.04 (<0.05).



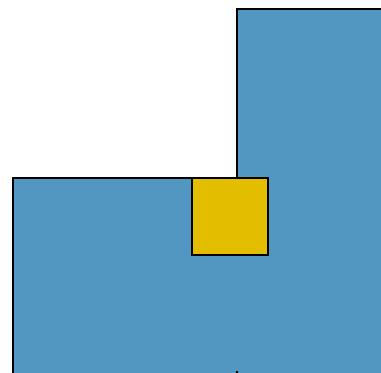
b) The constraint does not apply because the extension of the top edge is 0.01 (>0.009).

## Virtuoso Technology Data Constraint Reference

### Extension Constraints



c) The constraint does not apply to convex corners.



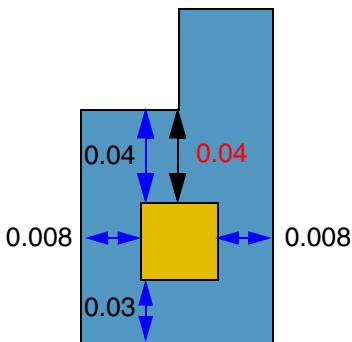
d) The constraint does not apply because the via cut touches the corner.

**Example 5: *minExtensionToCorner* with *toConcaveCorner*, *viaEdgeType*, and *exceptCornerTouch***

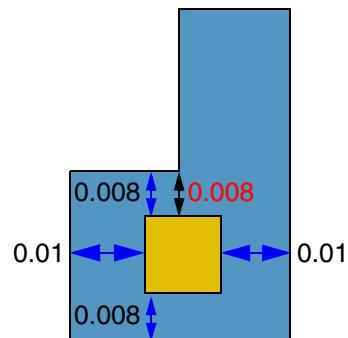
The distance of a Via1 via cut edge of viaEdgeType 1 from a concave corner of an enclosing Metal1 shape must be at least 0.05 (via cut edges with Metal1 extensions greater than 0.009 or a pair of opposite edges with extensions less than or equal to 0.009 are of viaEdgeType 1).

```
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 Constraint Reference

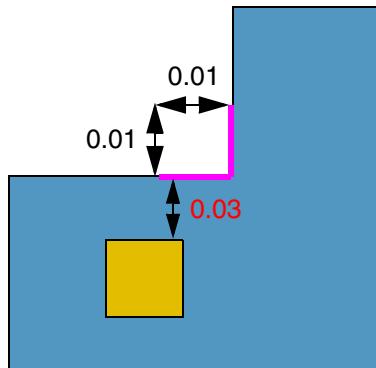
### Extension Constraints

#### ***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

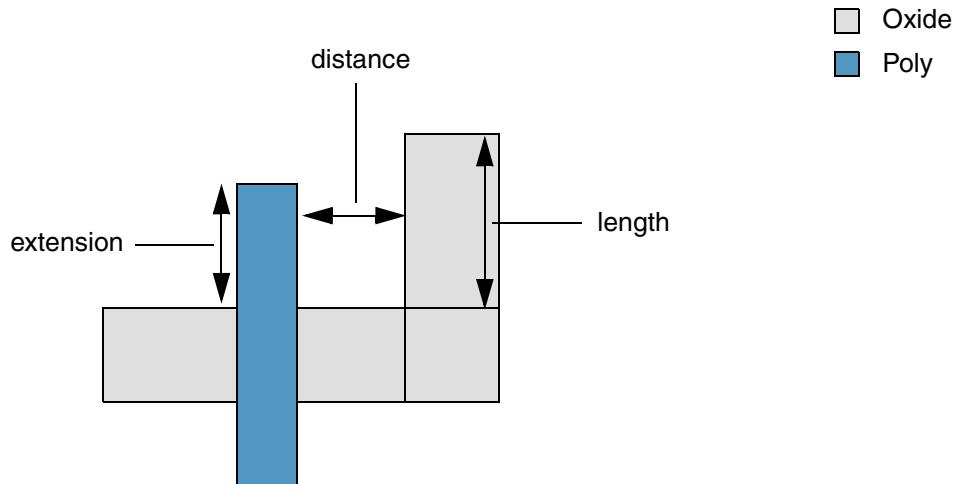


FAIL. The distance of the via cut from the concave corner (after the corner has been extended by 0.1) is 0.03 (<0.05).

## 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 Constraint 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 (Advanced Nodes Only)

```
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
```

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

'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 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](#)

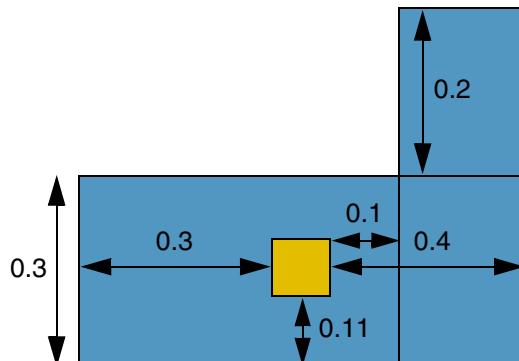
**Example 1: *minInnerVertexProximityExtension* with *cutClass*, *width*, *jogHeight*, *withinRange*, and *otherExtension***

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:

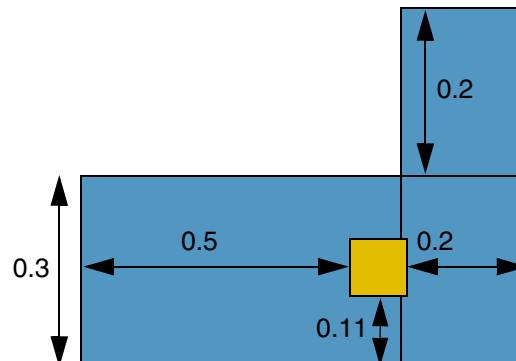
- Width of the enclosing Metal1 shape is less than or equal to 0.4
- Height of the jog is greater than 0.18
- Extension on the cut edge that is opposite to the inner vertex is less than 0.12

```
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 ( $\leq 0.4$ ); 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 ( $0.1 \leq \text{spacing} \leq 0.2$ ).

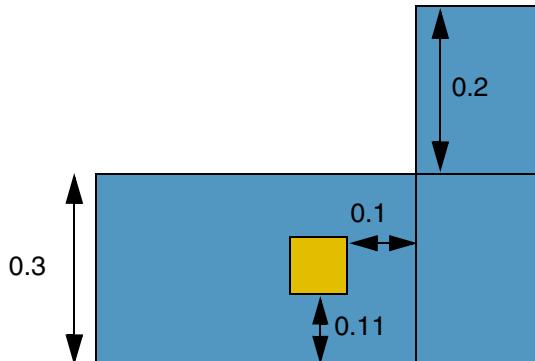
**Example 2: minInnerVertexProximityExtension with width, jogHeight, and otherExtension**

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:

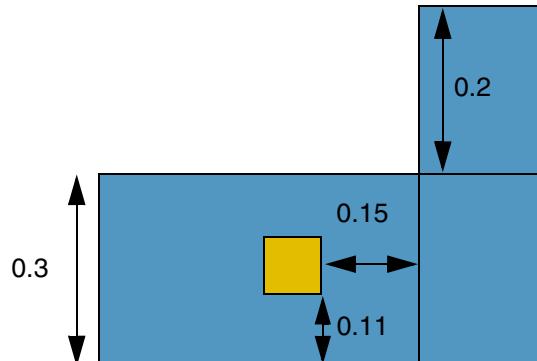
- Width of the enclosing Metal1 shape is less than or equal to 0.4
- Height of the jog is greater than 0.18
- Extension on the cut edge that is opposite to the inner vertex is less than 0.12

```
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 ( $\leq 0.4$ ); 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 ( $< 0.15$ ).

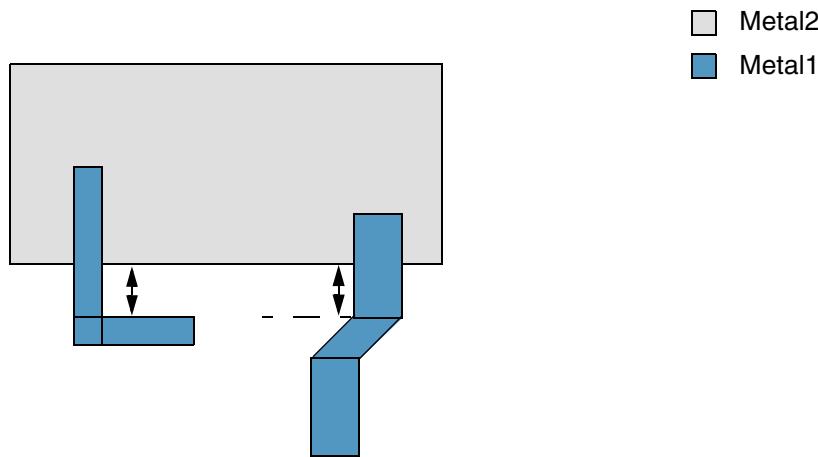


b) PASS. The distance of the via cut to the inner vertex must be 0.15, a condition that is met.

## **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 Constraint 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 (Advanced Nodes Only)

```
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
```

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

*tx\_metalLayer*      The metal layer on which the constraint is applied.

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

## Virtuoso Technology Data Constraint 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

'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 *meta1Layer*) 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 Constraint 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

(ICADV12.3 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](#)

**Example 1: minNeighborExtension with cutClass, paraLengthRange, withinRange, and width**

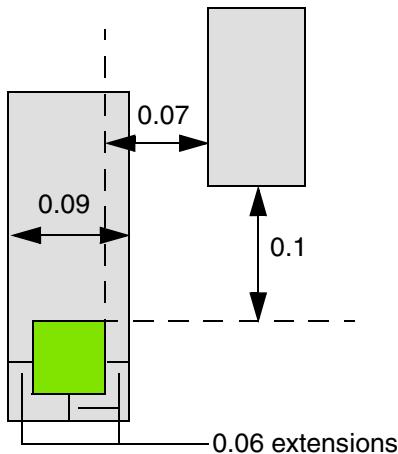
The extensions of a Metal2 shape on all sides of a Via2 cut shape with dimensions 0.03x0.03 must be at least 0.09 if a neighboring wire is present at a distance less than 0.08 from the cut shape and has parallel run length greater than or equal to -0.11 with the cut shape.

```

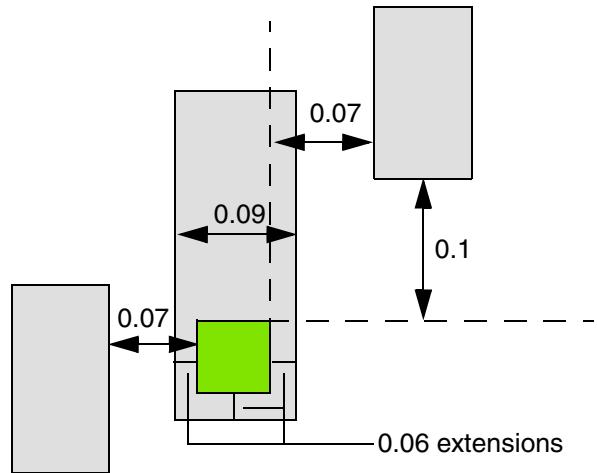
spacingTables(
  ( minOppExtension "Metal2" "Via2"
    ("width" nil nil)
    'cutClass 0.03
    (
      0.0 (0.05 0.0)
      0.1 (0.06 0.0)
    )
  )
) ;spacingTables
orderedSpacings(
  ( minNeighborExtension "Metal2" "Via2"
    'cutClass 0.03
    'paraLengthRange -0.11
    'withinRange 0.08
    'width 0.1
    (0.09 0.09)
  )
) ;orderedSpacings

```

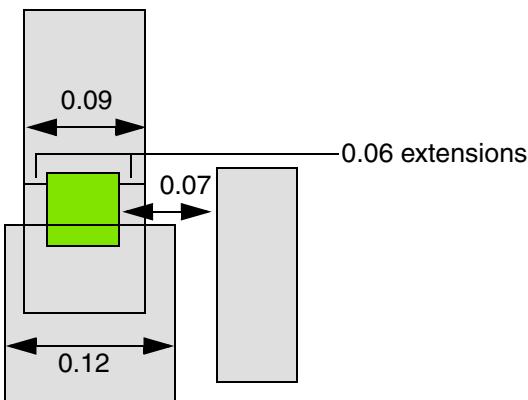
 Via2  
 Metal2



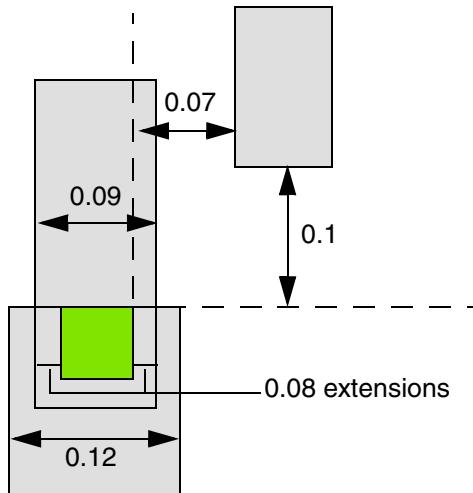
a) FAIL. The distance of the neighboring wire from the via cut is 0.07 (<0.08) and their parallel run length is -0.1 (>-0.11). However, Metal2 extensions on three sides are 0.06 (<0.09).



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.



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 Constraint Reference

### Extension Constraints

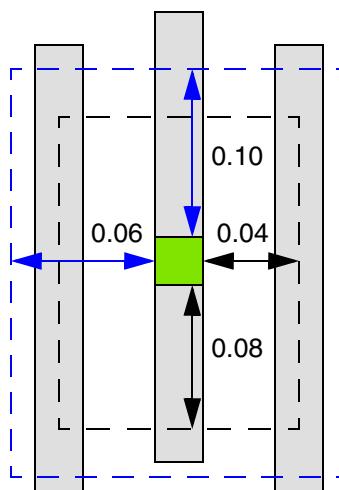
Because upper bounds are defined for parallel run length and distance (-0.1 and 0.06, respectively), the constraint does not apply if a neighboring wire is present on both sides of the cut shape inside the small search window.

```

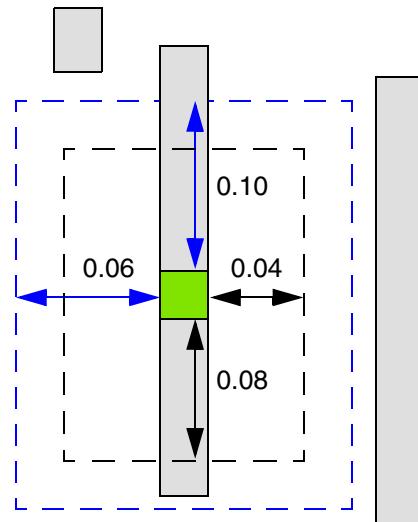
orderedSpacings(
    ( minOppExtension "Metal2" "Via2"
        'cutClass 0.03
        (0.02 0.0)
    )
) ;orderedSpacings
orderedSpacings(
    ( minNeighborExtension "Metal2" "Via2"
        'cutClass 0.03
        'paraLengthRange "[-0.08 -0.1]"
        'withinRange "[0.04 0.06]"
        (0.015 0.015)
    )
) ;orderedSpacings

```

█ Via2  
█ Metal2



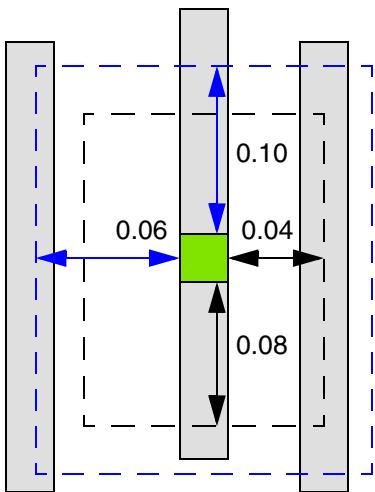
a) PASS. The `minNeighborExtension` constraint does not apply because neighboring wires are present inside the smaller search window on both sides of the via cut. The `minOppExtension` constraint applies and the extension requirement of (0.02 0.0) is met.



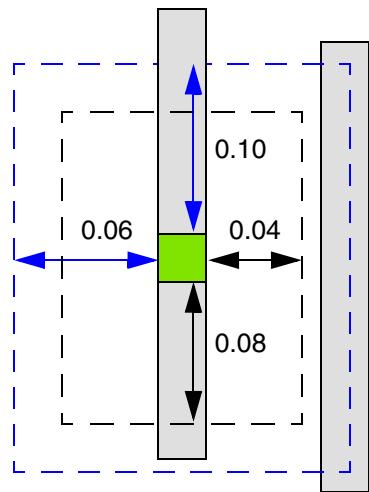
b) PASS. The `minNeighborExtension` constraint does not apply because a neighboring wire is not present inside either search window. The `minOppExtension` constraint applies and the extension requirement of (0.02 0.0) is met.

# Virtuoso Technology Data Constraint Reference

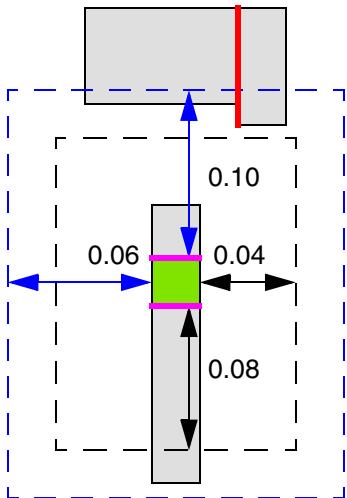
## Extension Constraints



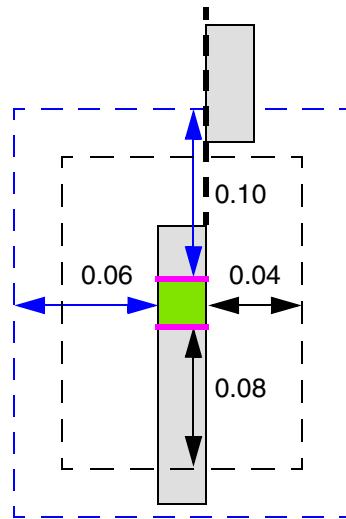
c) FAIL. A neighboring wire is present inside the small search window. However, the via cut does not meet the extension requirement of 0.015 on all four sides.



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 Constraint Reference

## Extension Constraints

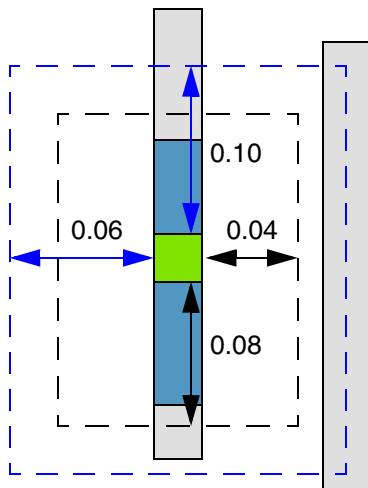
```

orderedSpacings(
    ( minOppExtension "Metal2" "Via2"
        'cutClass 0.03
        (0.02 0.0)
    )
) ;orderedSpacings

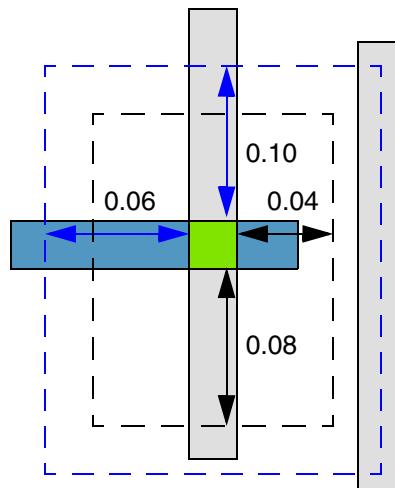
orderedSpacings(
    ( minNeighborExtension "Metal2" "Via2"
        'cutClass 0.03
        'paraLengthRange "[-0.08 -0.1]"
        'withinRange "[0.04 0.06]"
        'otherExtension 0.05
        'layer "Metal1"
        (0.015 0.015)
    )
) ;orderedSpacings

```

 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:

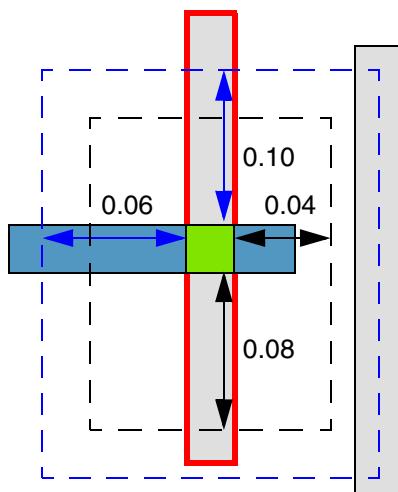
- The extension of the "below" metal (Metal1) shape is less than 0.05 on either side of the cut shape in a direction perpendicular to the direction of the "above" metal (Metal2) shape containing the cut shape.
- A neighboring wire is present:
  - On one side of the cut shape at a distance less than 0.04 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 Constraint 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

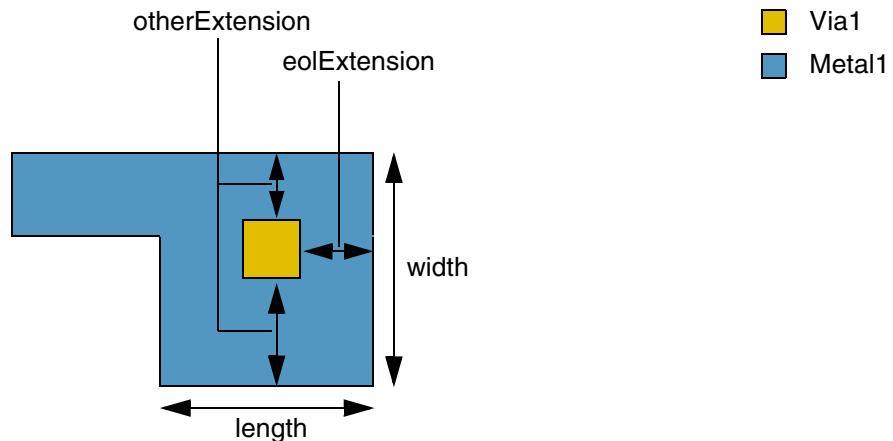
```

orderedSpacings(
    ( minOppEndOfLineExtension tx_layer1 tx_layer2
        'width f_width ['length f_length] ['eolOnly] ['shortEdgeOnEol]
        ['cutClass {f_width | (f_width f_length) | t_name}]
        ['endExclusion(f_endPerpendicularExcl f_endExcl)
         | 'sideExclusion (f_sideEdgeExcl f_backExcl f_fwdExcl)
        ]
        (f_eoleExtension 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:

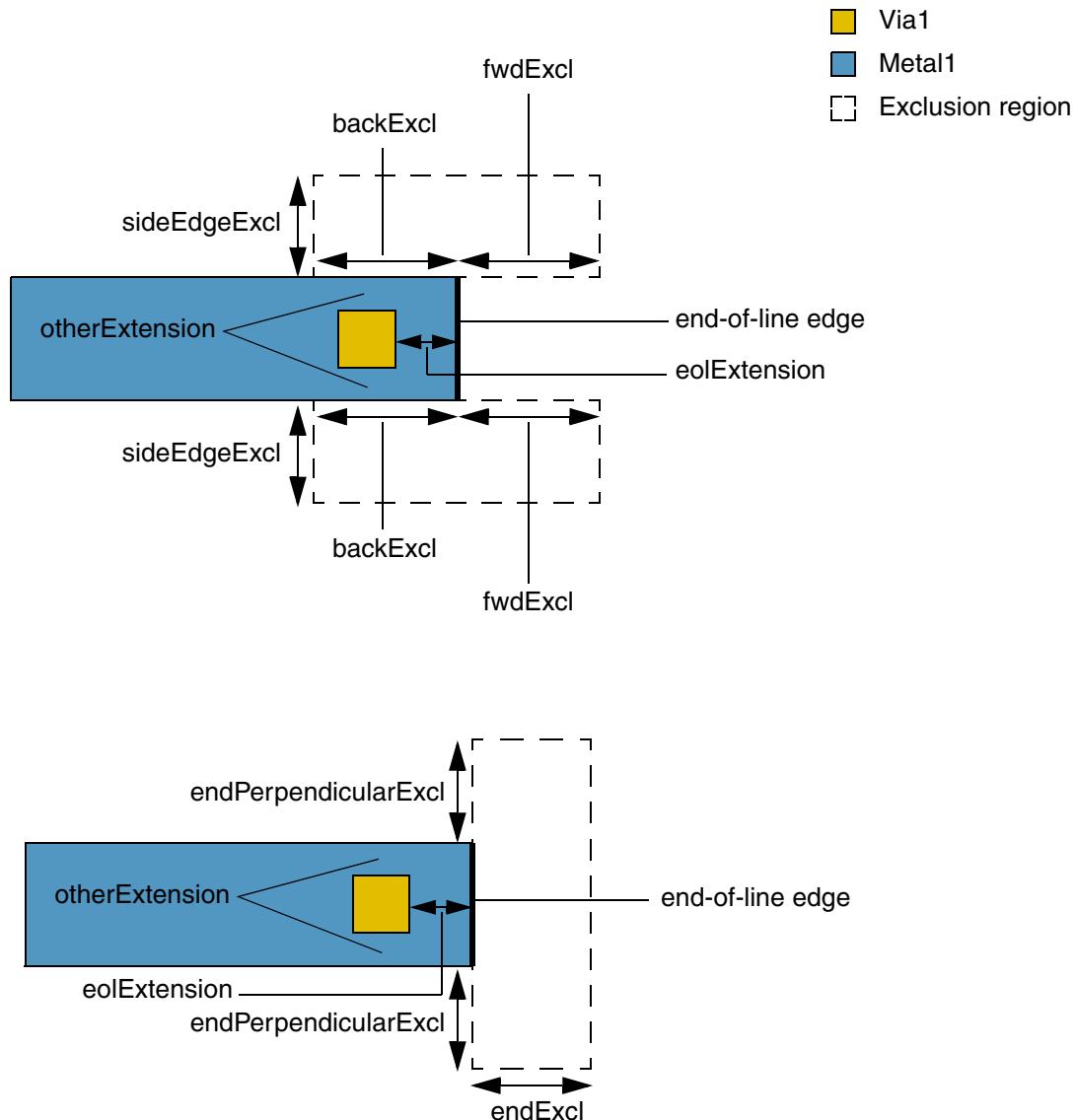


## Virtuoso Technology Data Constraint Reference

### 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 Constraint Reference

### Extension Constraints

---

#### Values

*tx\_layer1*      The first layer (metal) on which the constraint is applied.

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

*tx\_layer2*      The second layer (cut) on which the constraint is applied.

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

*f\_eolExtension f\_otherExtension*

The extension of the end-of-line edge and the edge opposite to it must be greater than or equal to *eolExtension* and the extension of the other pair of edges must be greater than or equal to *otherExtension*.

#### Parameters

'width *f\_width*      This constraint applies only if the width of the end-of-line edge is less than this value.

'length *f\_length*      This constraint applies only if the length of both edges adjoining an end-of-line edge with width less than *width* is greater than or equal to this value.

'eolOnly      Only `minOppEndOfLineExtension` constraints must be satisfied for all end-of-line edges. Other enclosure type constraints specified for the layer pair or cut class, such as `minOppExtension`, can be ignored.

If specified for one `minOppEndOfLineExtension` constraint, it must be specified for all `minOppEndOfLineExtension` 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 *eolExtension* and the long edges of the cut shape must have extension greater than or equal to *otherExtension*.

## Virtuoso Technology Data Constraint 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.

## Examples

- [Example 1: minOppEndOfLineExtension with width](#)
- [Example 2: minOppEndOfLineExtension with width and shortEdgeOnEol](#)
- [Example 3: minOppEndOfLineExtension with width, endExclusion, and sideExclusion](#)

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

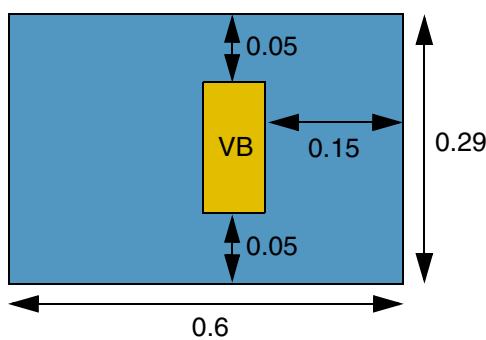
---

#### ***Example 1: minOppEndOfLineExtension with width***

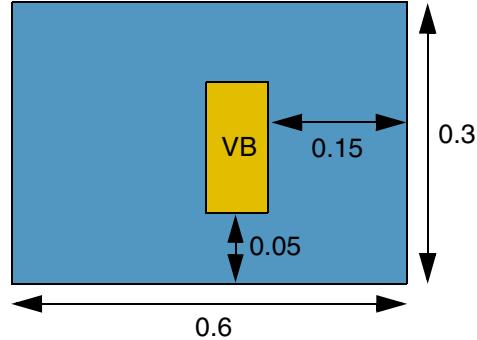
A VB via cut on layer Via1 must have an extension of at least 0.15 on an end-of-line edge and the opposite edge when the width of the enclosing Metal1 shape is less than 0.3; the extensions on the other two sides of the via cut must be at least 0.05.

```
orderedSpacings(
  ( minOppEndOfLineExtension "Metal1" "Via1"
    'width 0.3
    'cutClass "VB"
    (0.15 0.05)
  )
) ;orderedSpacings
```

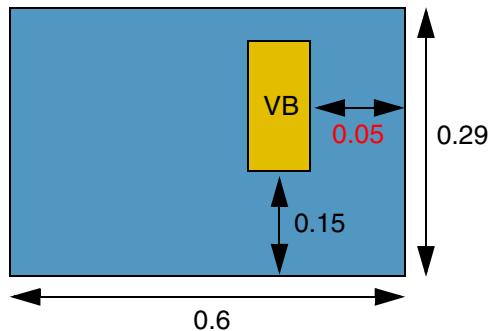
 Via1  
 Metal1



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  $\geq 0.30$ . 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 Constraint Reference

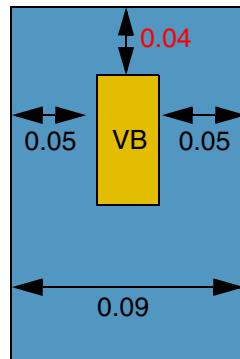
### Extension Constraints

#### ***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  
    '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 Constraint Reference

### Extension Constraints

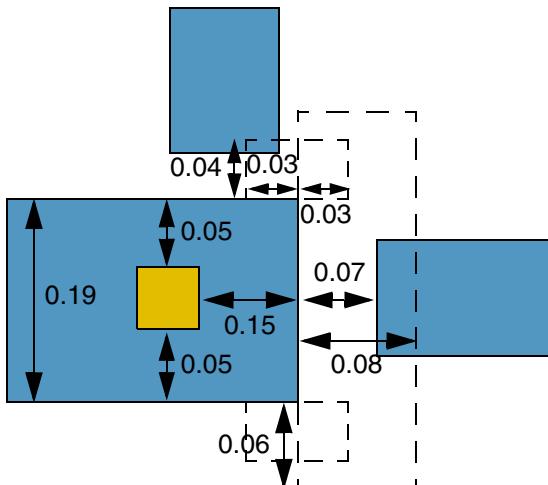
---

#### **Example 3: minOppEndOfLineExtension with width, endExclusion, and sideExclusion**

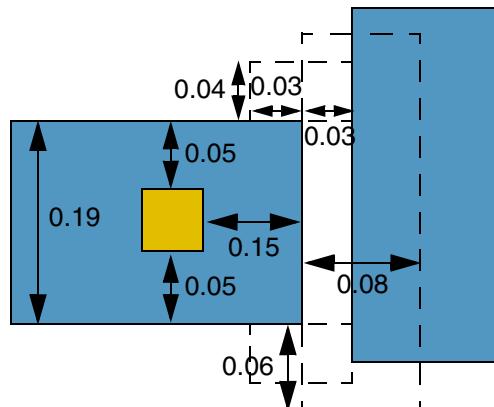
A via cut on layer Via1 must have extensions of at least 0.15 on an end-of-line edge and the opposite edge when the width of the enclosing Metal1 shape is less than 0.2; the extensions on the other two sides of the via cut must be at least 0.05. If a neighboring wire is present within 0.04, with forward 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
```

<span style="color: yellow;">█</span> Via1 <span style="color: blue;">█</span> Metal1 <span style="border: 1px solid black; padding: 2px;">□</span> 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.

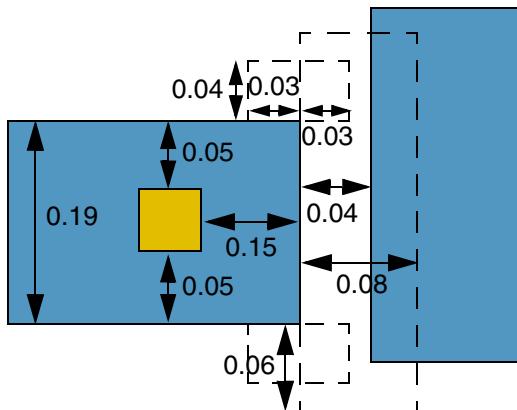


b) FAIL. The width of the end-of-line edge is 0.19 (<0.2). However, the extension requirement of (0.15, 0.05) does not apply because the neighboring wire is inside the end exclusion region and also touches the side exclusion region boundary. The extension requirement of (0.1, 0.1) applies, but is not met.

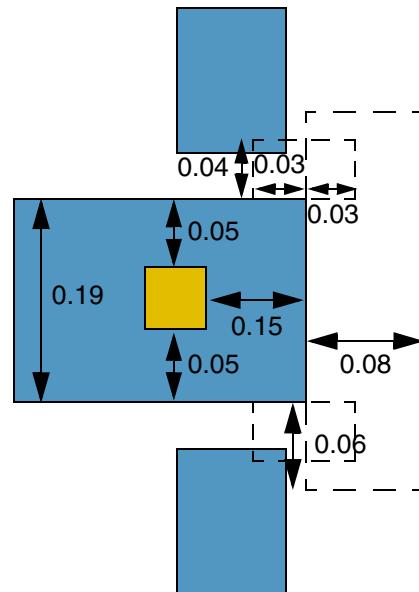
## Virtuoso Technology Data Constraint Reference

### 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.

## minOppExtension

```

orderedSpacings(
    ( minOppExtension tx_layer1 tx_layer2
        ['cutClass {f_width | (f_width f_length) | t_name}
         ['endSide]
        ]
        ['cutDistance f_cutDistance ['noSharedEdge | 'hasParaRunLength]]
        ['length f_length
            | 'extraCut ['extraOnly]
            | 'redundantCutWithin f_distance
        ]
        ['stepSizePair (f_val1 f_val2)]
        ['area f_area]
        ['horizontal | 'vertical]
        (f_ext f_oppExt) ['coincidentAllowed]
    )
)

; orderedSpacings

spacingTables(
    ( minOppExtension tx_layer1 tx_layer2
        (( "width" nil nil )
         ['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]
        ['stepSizePair (f_val1 f_val2)]
        ['horizontal | 'vertical]
        ['includeAbutted] ['minCorner]
        ['jogLengthOnly f_jogLength ['treatLAsJog]
            {'horizontalJog | 'verticalJog}
            'jogWireWidth f_jogWireWidth
        ]
        ['hollowHorizontal f_horiLength | 'hollowVertical f_vertLength]
        [f_default]
    )
    (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.

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

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.

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.

Generally, an OR constraint group must 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*.

*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.

## 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 Constraint Reference

### Extension Constraints

---

'hasExactPrl *f\_hasExactPrl*

(Advanced Nodes Only) 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 exactly 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

(ICADV12.3 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*

(Advanced Nodes Only) The constraint applies to a via with two or more cuts only if the cuts have parallel run length exactly 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 Constraint Reference

### Extension Constraints

---

'horizontal | 'vertical

(Advanced Nodes Only) 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

'exceptCutMetalEdgeExt

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.

In such a scenario, the cut shape must also meet the extension requirement that applies based on the width of the enclosing metal shape.

Type: Boolean

'includeAbutted

(Advanced Nodes Only) 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

'stepSizePair (f\_val1 f\_val2)

(Advanced Nodes Only) 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\*stepSizePair, where n>=0

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

'jogLengthOnly *f\_jogLength*

(ICADV12.3 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

(ICADV12.3 Only) The constraint also applies to an L-shaped jog.

Type: Boolean

'horizontalJog | 'verticalJog

(ICADV12.3 Only) The direction in which the length of the jog is measured.

Type: Boolean

'jogWireWidth *f\_jogWireWidth*

(ICADV12.3 Only) The constraint applies only if the width of a Z-shaped jog is equal to this value.

'hollowHorizontal *f\_horiLength* | 'hollowVertical *f\_vertLength*

(ICADV12.3 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.

*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

## 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](#)

## Virtuoso Technology Data Constraint Reference

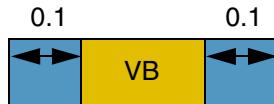
### Extension Constraints

#### **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.



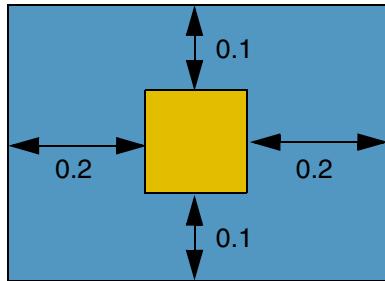
b) FAIL. The extension must be 0.1 on the short edges and zero on the long edges.

**Example 2: minOppExtension with vertical**

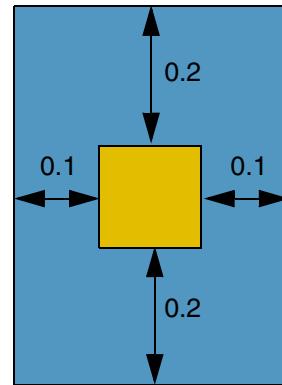
A Via1 via cut enclosed by a Metal1 wire must have 0.1 extensions in the vertical direction and 0.2 extensions in the horizontal direction.

```
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.



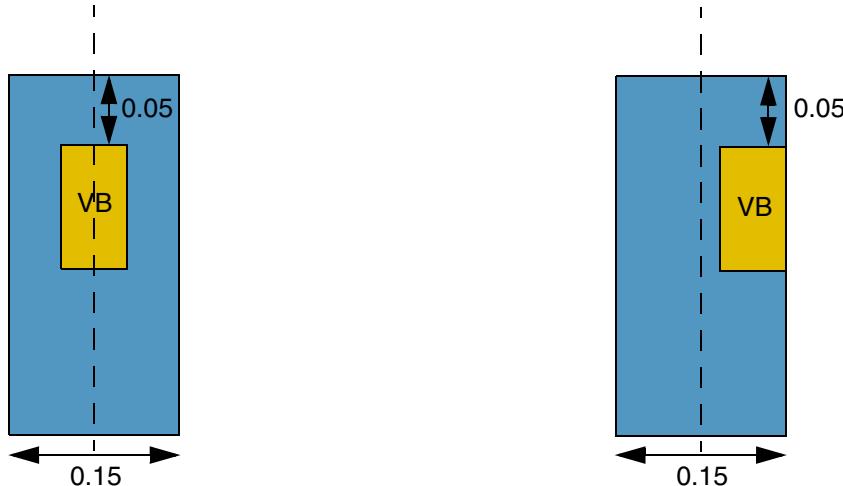
b) FAIL. The extensions in the vertical direction must be 0.1. The constraint would pass if the direction was not specified.

**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" "Vial"
        (( "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 exactly 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 Constraint Reference

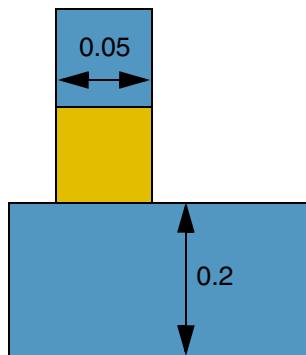
### Extension Constraints

#### **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



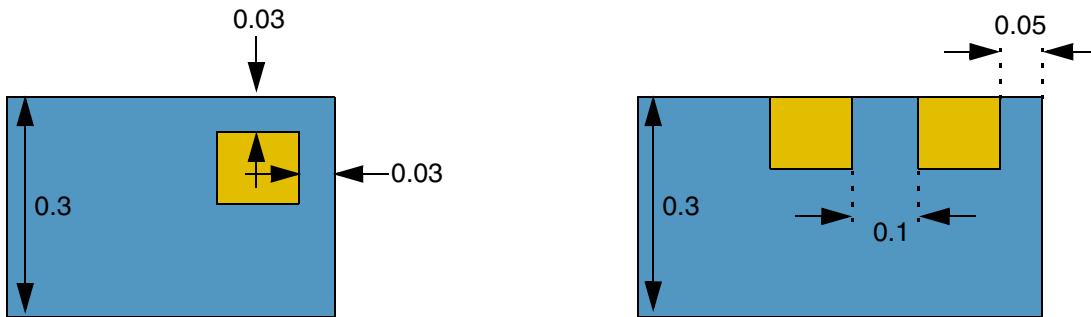
FAIL. The via cut is abutted to a wide wire, but does not have the required 0.05 extensions on all sides. The constraint would not apply if 'includeAbutted' was not specified.

**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.

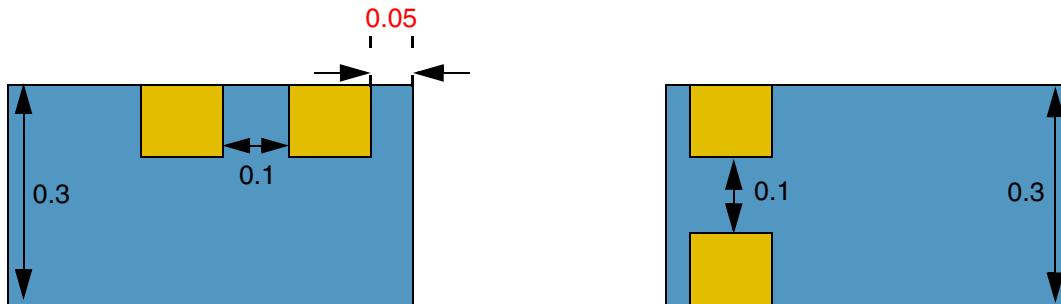
b) The constraint does not apply because the distance between the via cuts is 0.01 (<0.02).

**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 Constraint Reference

### Extension Constraints

---

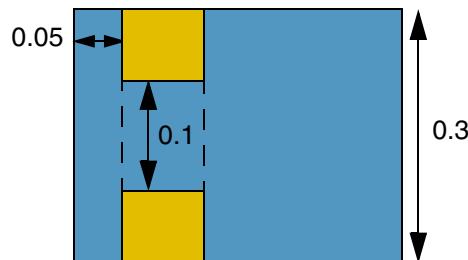
When the width-based constraint does not apply, an extension requirement of (0.0, 0.05) or (0.2, 0.2) must be met.

```

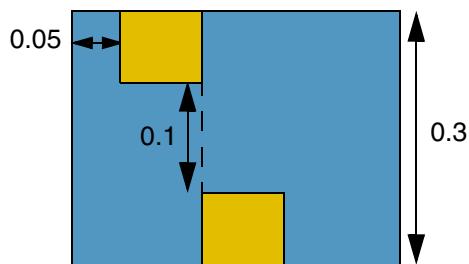
orderedSpacings(
    ( minOppExtension "Metall1" "Vial"
        (0.0 0.5)
    )
    ( minOppExtension "Metall1" "Vial"
        (0.2 0.2)
    )
) ;orderedSpacings

spacingTables(
    ( minOppExtension "Metall1" "Vial"
        (( "width" nil nil )
         'cutDistance 0.2 'hasParaRunLength
        )
        (
            0.30 (0.03 0.03)
        )
    )
) ;spacingTables

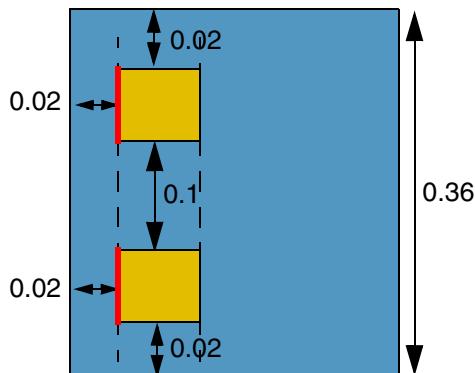
```



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.



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***

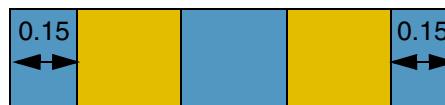
If two or more Via1 via cuts overlap a contiguous Metal1 shape, the via cuts must have 0.0 extension in one direction and 0.15 extension in the perpendicular direction.

```
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.

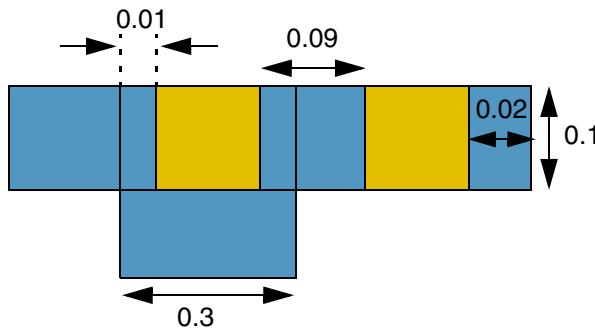
```

orderedSpacings(
    ( minOppExtension "Metal1" "Via1"
        'extraCut
        ( 0.00 0.02)
    )
    ( minOppExtension "Metal1" "Via1"
        'redundantCutWithin 0.1
        ( 0.00 0.01)
    )
)

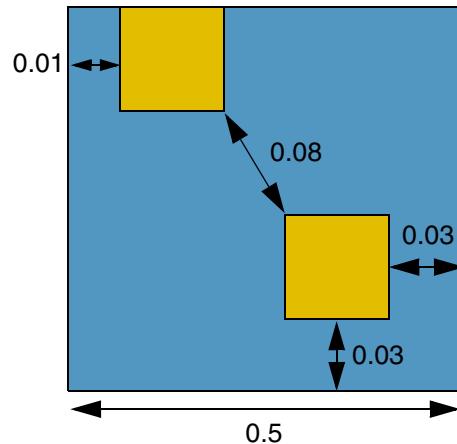
) ; orderedSpacings
spacingTables(
    ( minOppExtension "Metal1" "Via1"
        (( "width" nil nil )
        )
        ( 0.3 ( 0.03 0.03 ) )
    )
) ; spacingTables

```

 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***

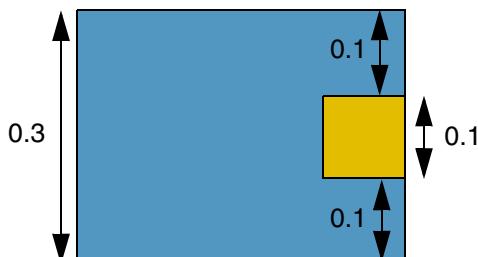
If an edge of a Via1 via cut or an edge of a Metal1 shape enclosing the via cut protrudes over a wide wire, it is exempted from the extension requirement. The following are the extension requirements based on width:

- 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.

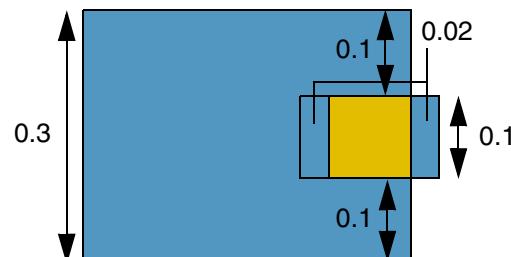
```

orderedSpacings(
    ( minOppExtension "Metall1" "Via1"
        ( 0.00 0.02)
    )
) ;orderedSpacings
spacingTables(
    ( minOppExtension "Metall1" "Via1"
        ( ( "width" nil nil )
            'exceptCutMetalEdgeExt
        )
        (
            0.15 (0.02 0.02)
            0.30 (0.03 0.03)
        )
    )
) ;spacingTables

```



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.

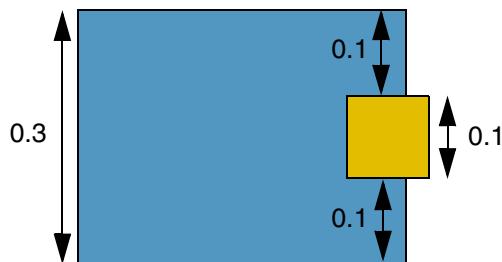


b) PASS. The protrusion on the right side exempts that edge of the via cut, and the other three edges have 0.03 extensions. Basic extension requirement of (0.0, 0.02) is also met.

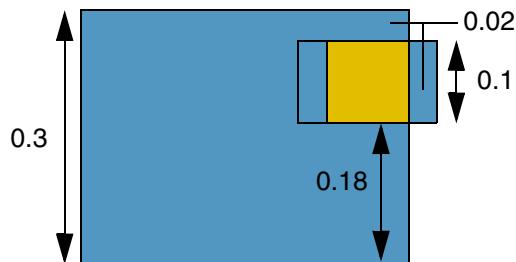
## Virtuoso Technology Data Constraint Reference

### Extension Constraints

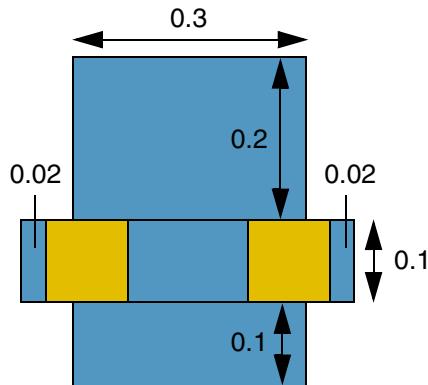
---



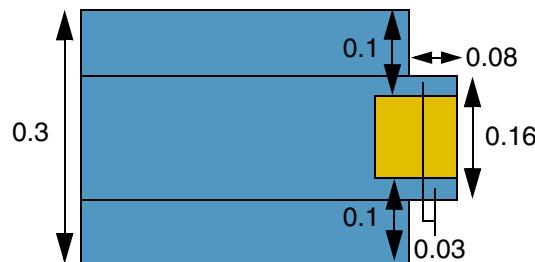
c) FAIL. The protrusion on the right side exempts that edge of the cut shape, and the other three edges have 0.03 extensions. However, the basic extension requirement of (0.0, 0.02) is not met.



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 Constraint Reference

### Extension Constraints

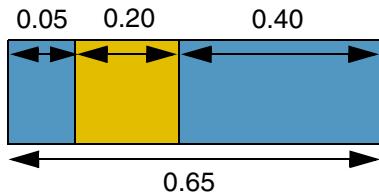
---

#### **Example 11: minOppExtension with length**

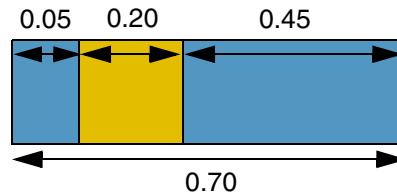
A Via1 via cut enclosed by a Metal1 wire with length greater than or equal to 0.70 must have 0.05 extension in one direction and 0.00 extension in the perpendicular direction.

```
orderedSpacings(
    ( minOppExtension "Metal1" "Via1"
        'length 0.70
        (0.05 0.00)
    )
) ;orderedSpacings
```

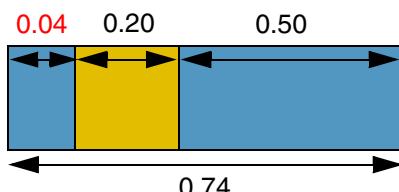
 Via1  
 Metal1



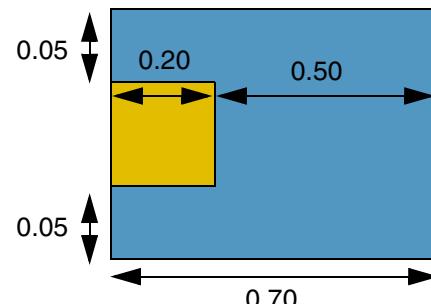
a) The constraint does not apply because the length of the wire is 0.65 (<0.70).



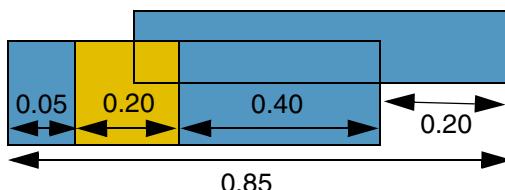
b) PASS. The length of the wire is 0.70, and the horizontal extension is 0.05 and the vertical extension is 0.



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 Constraint Reference

### Extension Constraints

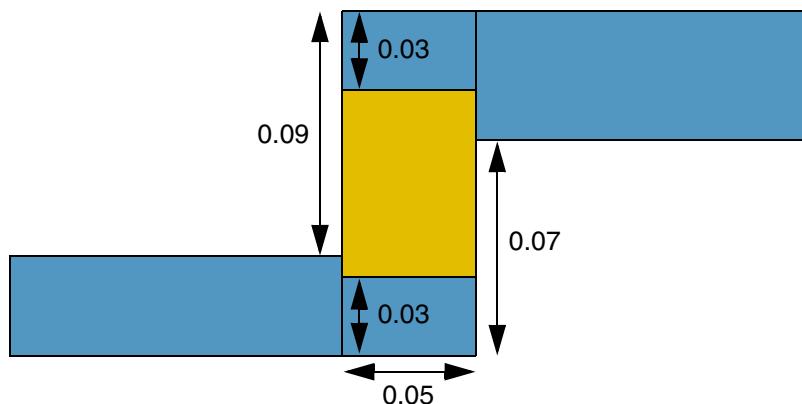
#### **Example 12: minOppExtension with jogLengthOnly, verticalJog, and jogWireWidth**

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.

## Virtuoso Technology Data Constraint Reference

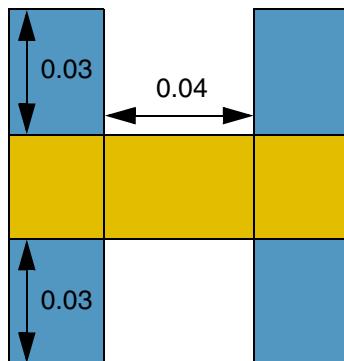
### Extension Constraints

#### **Example 13: minOppExtension with hollowHorizontal**

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 does not require an extension.

```
spacingTables(  
  ( minOppExtension "Metal1" "Via1"  
    ( ( "width" nil nil )  
      'hollowHorizontal 0.02  
    )  
    ( 0.00 ((0.00 0.03))  
    )  
  ) ;spacingTables
```

 Via1  
 Metal1

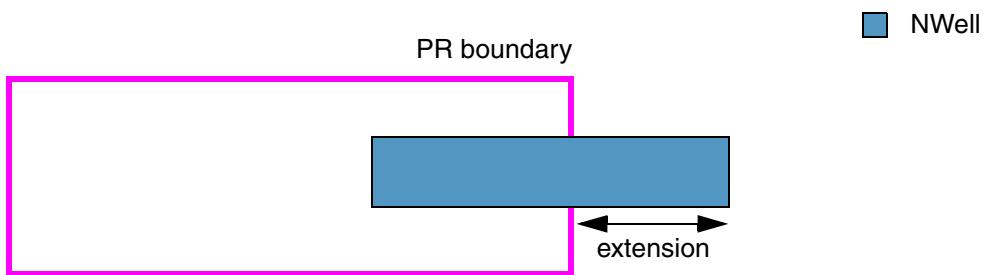


PASS. The middle portion of the via cut with width 0.04 (the center of the via cut extended horizontally by 0.02 on both sides) does not require an extension.

## 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 Constraint 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 (Advanced Nodes Only)**

```
orderedSpacings(
  ( minProtrusionWidthWithVia tx_metalLayer tx_cutLayer
    'widthLengthTable ((f_wideWidth f_length) ...)
    'cutClass {f_width | (f_width f_length) | t_name}
    f_width
  )
) ;orderedSpacings
```

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.

## 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:

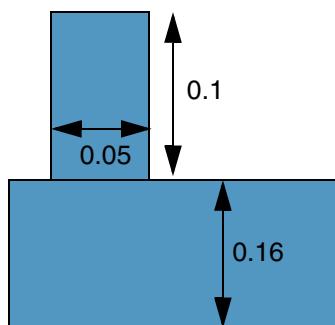
- The width of the protrusion must be at least 0.05.
- If the width of the wide wire is greater than or equal to 0.11, the length of the protrusion must be at least 0.12.

# Virtuoso Technology Data Constraint Reference

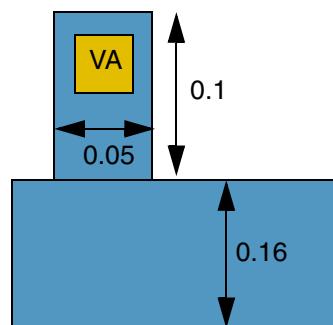
## Extension Constraints

```
orderedSpacings(  
    ( minProtrusionWidthWithVia "Metal1" "Via1"  
        'widthLengthTable ( 0.11 0.12 )  
        'cutClass "VA"  
        0.05  
    )  
) ;orderedSpacings
```

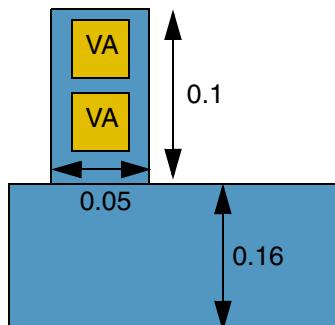
 Via1  
 Metal1



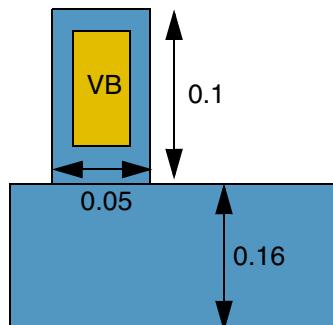
a) The constraint does not apply because the protrusion does not enclose a via cut.



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 | 'withinFirstWidth]]
     ['horizontal | 'vertical]
     ['mask1 | 'mask2 | 'mask3]
     ['directionalExtension]
     ['otherLayer tx_otherLayer ['parallelWithinTable g_paraWithinTable]
      |
      'trimLayer tx_trimLayer ['trimLengthTable g_trimLengthTable]
     ]
     ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
     ]
     [f_default]
   )
   (g_table)
 )
)
;spacingTables
```

Specifies the minimum extension of a shape with the specified minimum width on *layer1* past a shape on *layer2*. Typically, *layer1* is a metal layer and *layer2* is a cut layer. 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.

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.

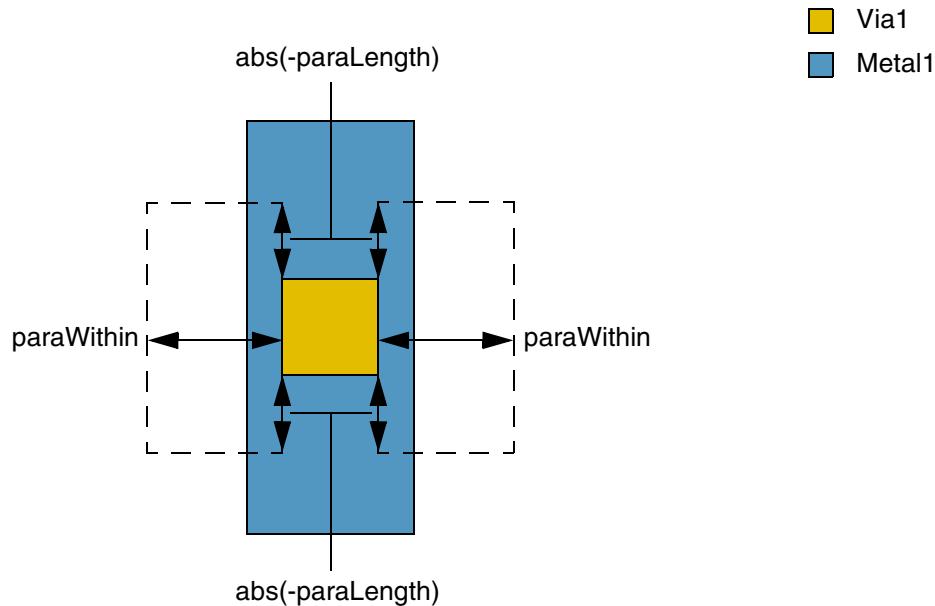
The `minQuadrupleExtension` constraint can be set to apply only in a certain direction. In this case, the extensions in the specified direction must meet the first constraint-value-pair, and the extensions in the perpendicular direction must meet the second constraint-value-pair.

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.

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

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).



Generally, an OR constraint group must 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

<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> . The <i>table</i> row is defined as follows:  <pre>( f_width ((f_ext1 f_ext2 f_ext3 f_ext4 ['minSum] ['maxLength g_maxLength]) ... )</pre> where, <ul style="list-style-type: none"><li>■ <i>f_width</i> is the width of the <i>layer1</i> shape. The constraint applies only if the width of the <i>layer1</i> shape is greater than or equal to this value.</li><li>■ <i>f_ext1 f_ext2 f_ext3 f_ext4</i> are extension pairs. The first pair specifies <i>layer1</i> extensions on one set of opposite sides/edges of the <i>layer2</i> shape, and the second pair specifies extensions on the other set of opposite sides/edges of the <i>layer2</i> shape.</li></ul> 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 <i>f_maxLength</i> : (ICADV12.3 Only) The constraint applies only if the length of the <i>layer1</i> shape is less than or equal to this value.

## Virtuoso Technology Data Constraint 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

(ICADV12.3 Only) The width of the widest metal shape that the cut shape overlaps is used to look up the extension requirement.

Type: Boolean

'withinFirstWidth

(ICADV12.3 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

'mask1 | 'mask2 | 'mask3

(ICADV12.3 Only) The constraint applies only to the shapes on the specified mask.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

'directionalExtension

(ICADV12.3 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

(ICADV12.3 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)

(ICADV12.3 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 actual 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.

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

`'trimLayer tx_trimLayer`

(ICADV12.3 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)

`'trimLengthTable g_trimLengthTable`

(ICADV12.3 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 6: minQuadrupleExtension with trimLayer and trimLengthTable](#).

Type: A 1-D table indexed on width, specifying corresponding floating-point length value for the trim shape.

## Virtuoso Technology Data Constraint Reference

### Extension Constraints

---

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

(ICADV12.3 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 (*tx\_layer1 tx\_layer2 ... tx\_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 (*tx\_layer1 tx\_layer2 ... tx\_layerN*).

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

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

*f\_default*

The extension value to be used when no table entry applies.

## Examples

- [Example 1: minQuadrupleExtension with horizontal](#)
- [Example 2: minQuadrupleExtension with allSides and minSum](#)
- [Example 3: minQuadrupleExtension with otherLayer](#)
- [Example 4: minQuadrupleExtension with maxLength](#)
- [Example 5: minQuadrupleExtension with allSides and withinFirstWidth](#)
- [Example 6: minQuadrupleExtension with trimLayer and trimLengthTable](#)
- [Example 7: minQuadrupleExtension in an OR group](#)

## Virtuoso Technology Data Constraint Reference

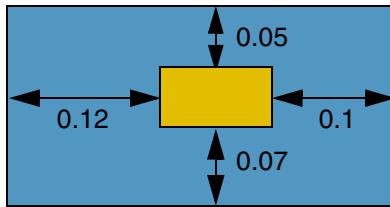
### Extension Constraints

#### ***Example 1: minQuadrupleExtension with horizontal***

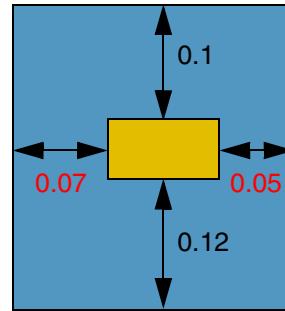
The extensions must be 0.1 and 0.12, measured in the horizontal direction, and 0.05 and 0.07, measured in the vertical direction.

```
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 Constraint Reference

### Extension Constraints

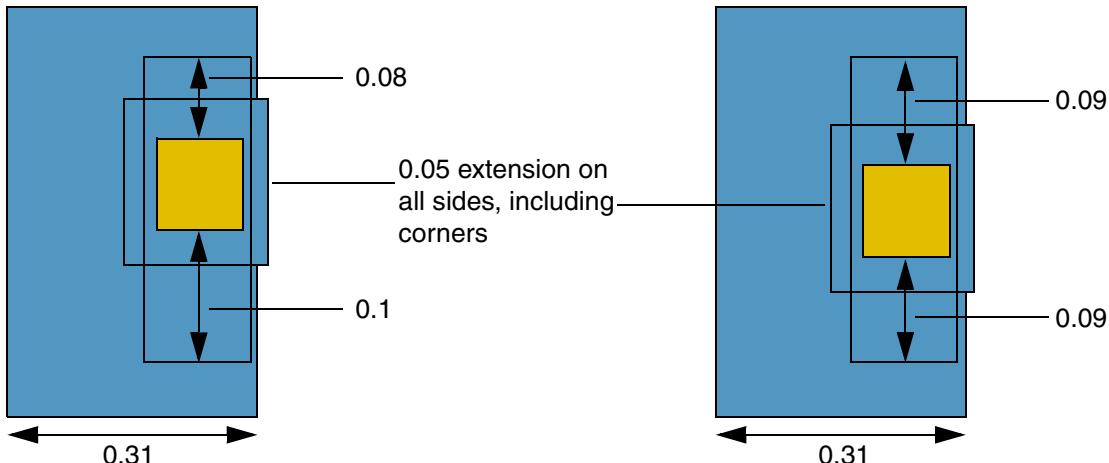
---

#### ***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 Constraint Reference

### Extension Constraints

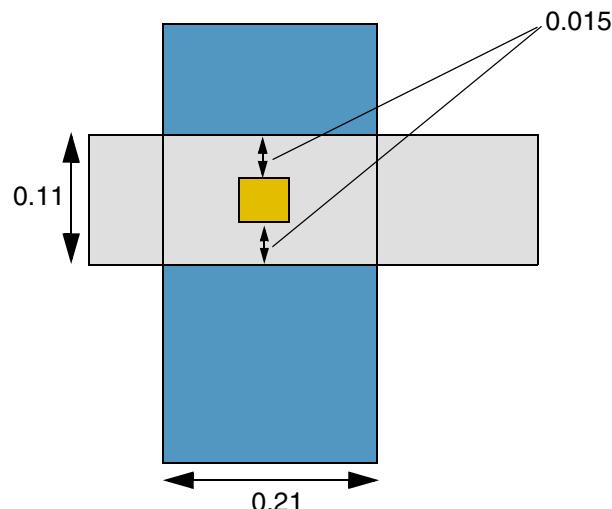
#### ***Example 3: minQuadrupleExtension with otherLayer***

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.

```
spacingsTables(
  ( 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 Constraint Reference

### Extension Constraints

---

#### ***Example 4: 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.

```

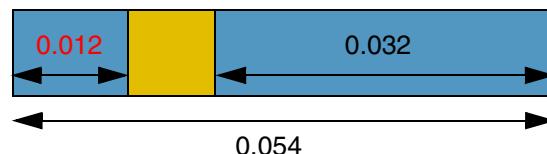
spacingTables(
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    ( 0.0 ((0.0 0.0 0.015 0.015)) )
  )
  ( minQuadrupleExtension "Metal1" "Via1"
    (( "width" nil nil ))
    ( 0.0 ((0.0 0.0 0.012 0.012 'maxLength 0.05)) )
  )
) ;spacingTables

```





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 Constraint Reference

### Extension Constraints

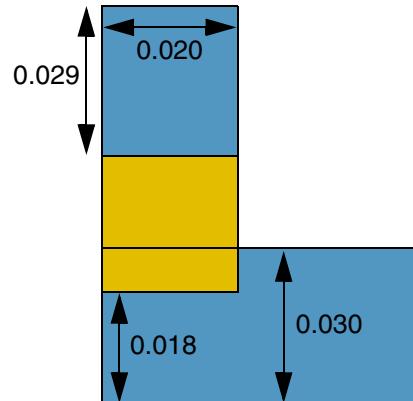
#### ***Example 5: minQuadrupleExtension with allSides and withinFirstWidth***

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 Constraint Reference

### Extension Constraints

---

#### **Example 6: minQuadrupleExtension with trimLayer and trimLengthTable**

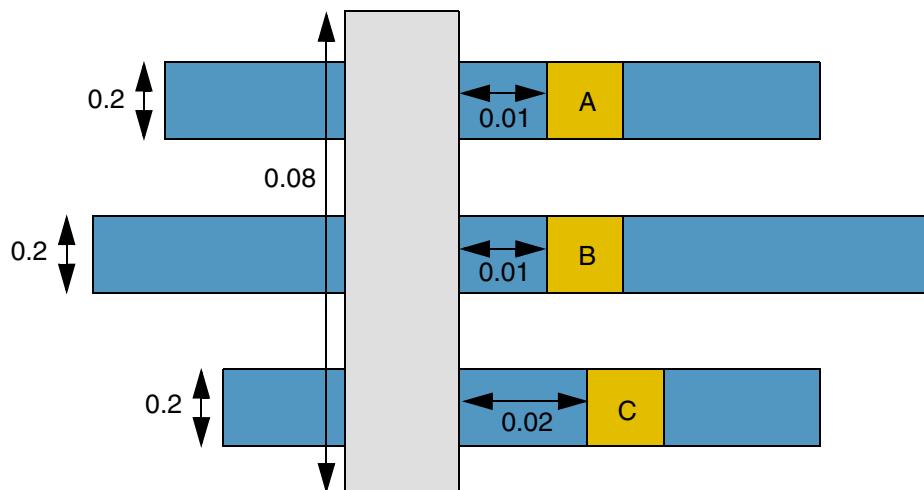
A Metal1 wire, which is overlapped by a TrimMetal1 shape with length greater than or equal to 0.08, must satisfy a minimum extension requirement of 0.02 on two opposite sides of a Via1 via shape. The check applies only to the two outermost wires in the given set.

```

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 ( $\geq 0.08$ ), 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.

***Example 7: 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

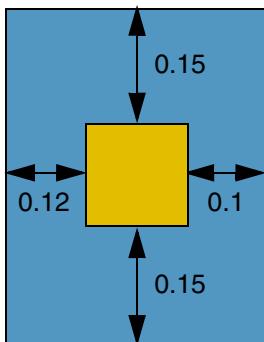
# Virtuoso Technology Data Constraint Reference

## Extension Constraints

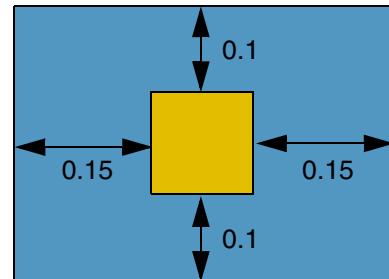
```

constraintGroups(
  ( "extensionGroup" nil nil 'or
    orderedSpacings(
      ( minExtensionDistance "Metal1" "Vial1" 'horizontal 0.2 )
      ( minOppExtension "Metal1" "Vial1" 'horizontal (0.15 0.1) )
    ) ;orderedSpacings
    spacingTables(
      ( minQuadrupleExtension "Metal1" "Vial1"
        ( ( "width" nil nil )
          'horizontal
        )
        ( 0 ((0.12 0.1 0.15 0.15)) )
      )
    ) ;spacingTables
  ) ;extensionGroup
  ( "foundry" nil
    memberConstraintGroups ("extensionGroup")
  ) ;foundry
) ;constraintGroups

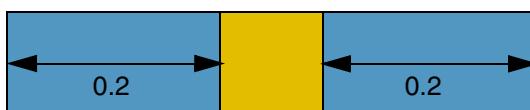
```



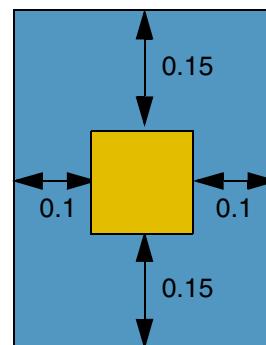
a) PASS. The `minQuadrupleExtension` constraint is met.



b) PASS. The `minOppExtension` constraint is met.



c) PASS. The `minExtensionDistance` constraint is met.



d) FAIL. None of the constraints is met.

## 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

'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*

(Advanced Nodes Only) 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 Constraint Reference

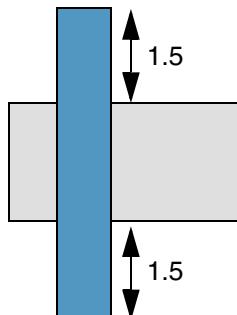
### Extension Constraints

#### **Example 1: minSideExtension**

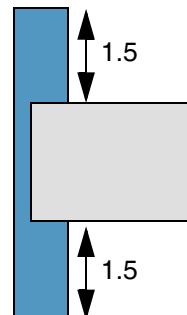
The extension of a Metal1 shape on two opposite sides of a Metal2 shape must be at least 1.5.

```
orderedSpacings(  
    ( minSideExtension "Metal1" "Metal2"  
        1.5  
    )  
) ;orderedSpacings
```

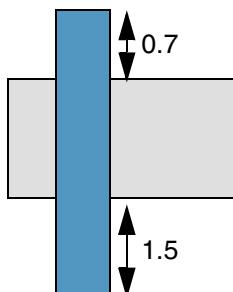
 Metal2  
 Metal1



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 Constraint Reference

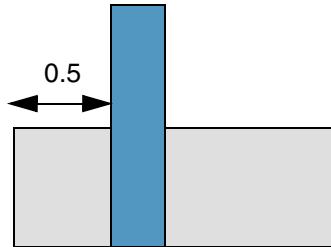
### Extension Constraints

#### **Example 2: minSideExtension with otherExtension**

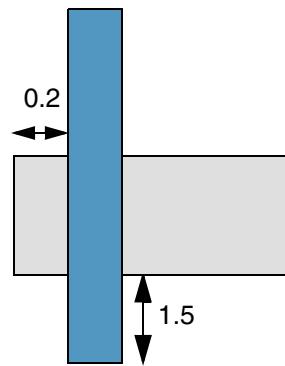
The extension of a Metal1 shape on a pair of opposite sides of a Metal2 shape must be at least 2.0 if the extension of one side of the Metal2 shape past the Metal1 shape is less than or equal to 0.2.

```
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).



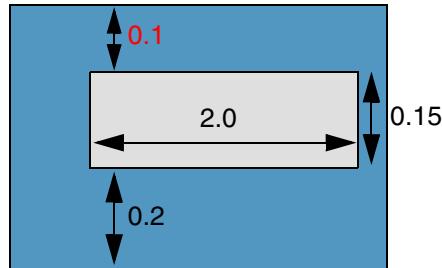
- a) FAIL. The extension of the Metal2 shape past a side of the Metal1 shape is 0.2, but the extension of the Metal1 shape past the Metal2 shape is only 1.5 (<2.0).

**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, exactly equal to 0.5, or exactly equal to 0.6.

```
orderedSpacings(  
  ( minSideExtension "Metal1" "Metal2"  
    'longEdge  
    'exceptEdgeLengthRanges ("<=0.2" "0.5" "0.6")  
    0.12  
  )  
) ; orderedSpacings
```

 Metal2  
 Metal1



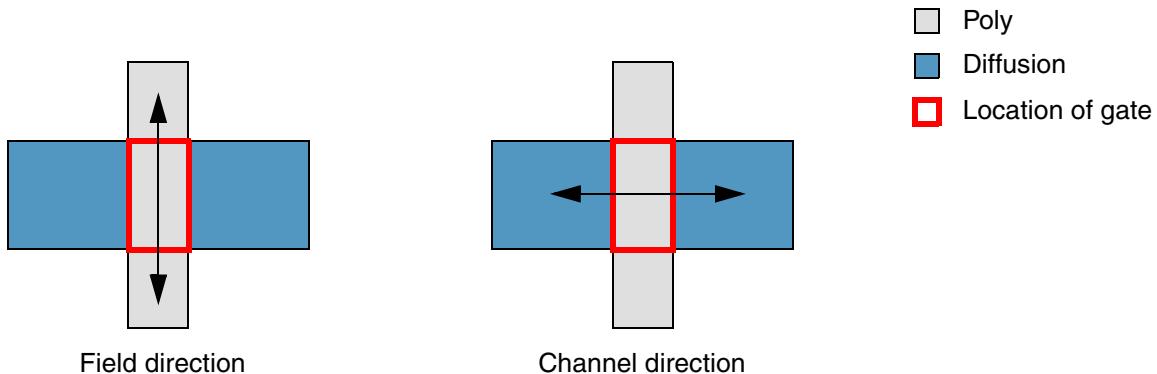
FAIL. The constraint applies because the length of the Metal2 shape does not fall in the exempted length ranges (but applies only to the top and bottom Metal1 edges because 'longEdge is specified). However, the extension of the top edge of the Metal1 shape past the Metal2 shape is only 0.1 (<0.12).

## **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.

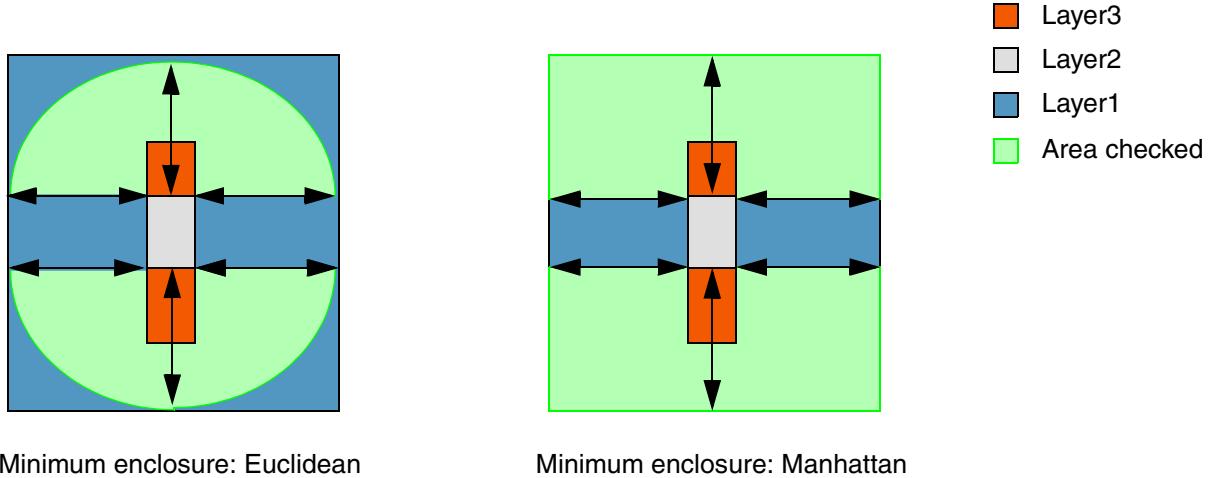


**Note:** Generally, the `minTouchingDirEnclosure` constraint is used to check the overlap of "Poly" over "Diffusion".

## Virtuoso Technology Data Constraint 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. 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.

## 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

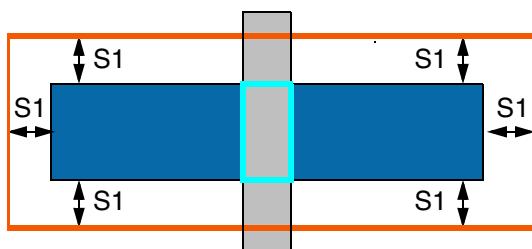
The `minEnclosure` and `minTouchingDirEnclosure` constraints are used with a derived layer named `polyAndDiffusion` (equivalent to `layer3`) to create a larger minimum implant enclosure around the `poly` endcap, than is applied around the rest of the diffusion.

```
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 Constraint Reference

## Extension Constraints

Minimum enclosure of diffusion by implant specified by constraint `minEnclosure layer1 (implant) enclosing layer2 (diffusion) = 1.0`



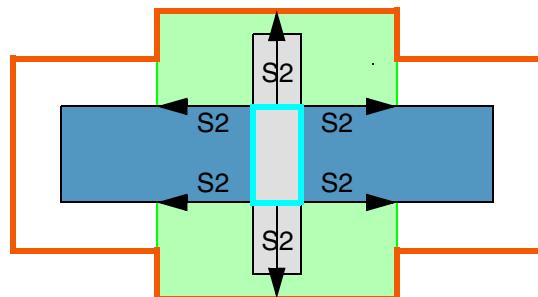
- poly
- diffusion
- polyAndDiffusion
- implant
- Area checked

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>	

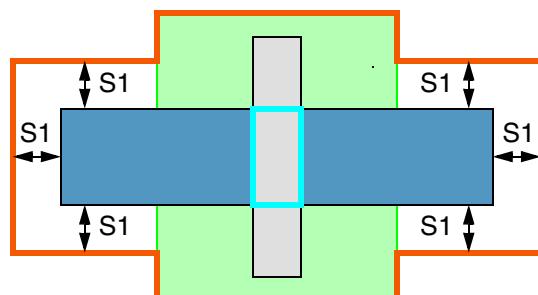
`minTouchingDirEnclosure` applies:

- 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

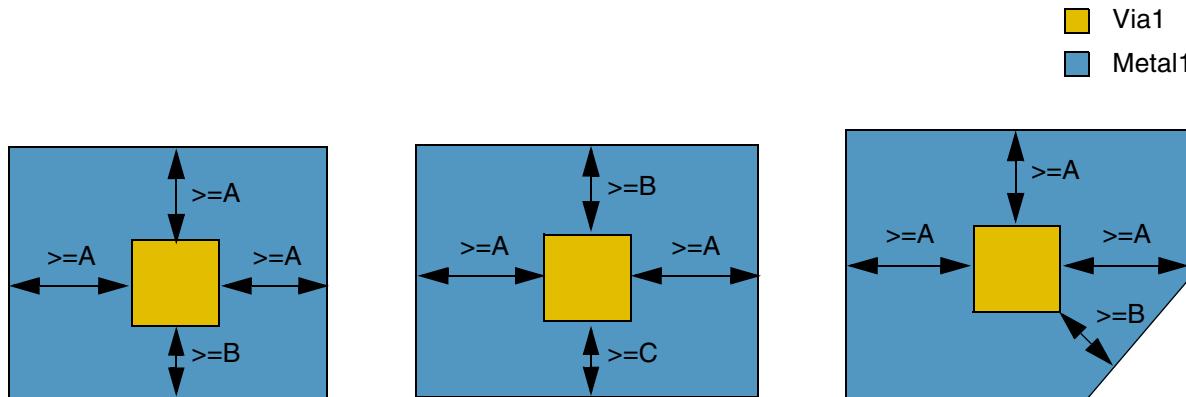
`minEnclosure of layer1 enclosing layer2` applies to the remaining areas of implant over diffusion



## 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)

## 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

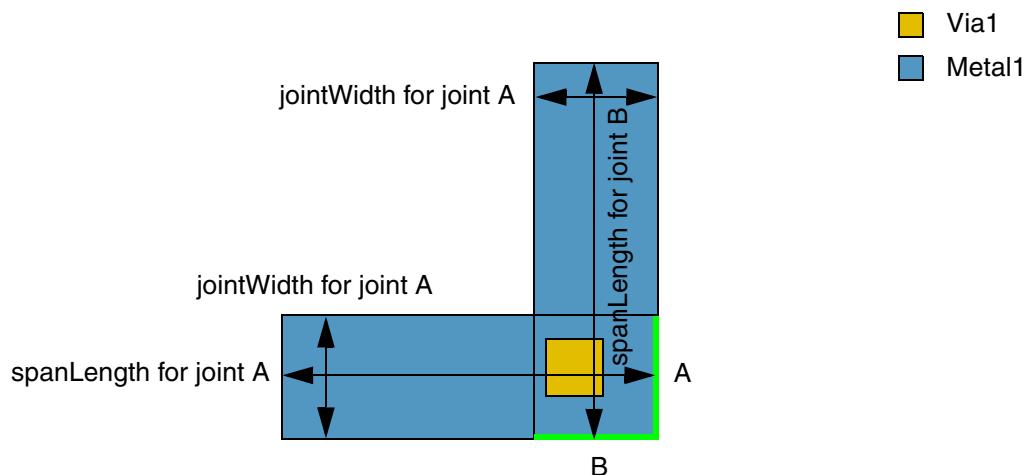
```
orderedSpacings(  
  ( minViaExtension "Metal1" "Via1" 'oneOneTwo (0.2 0.3 0.4)  
  )  
) ; orderedSpacings
```

## 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 Constraint 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>   ( <i>f_extension1 f_extension2</i> )	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

'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

'jointWidth *f\_jointWidth*

The constraint applies only if the width of a joint is less than this value.

'spanLength *f\_spanLength*

The constraint applies only if the span length of a joint is greater than this value.

'eolMinLength *f\_eolMinLength*

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 Constraint Reference

### Extension Constraints

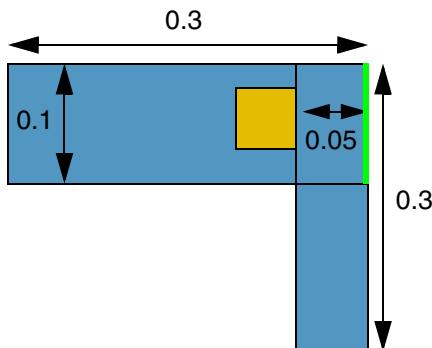
---

#### Example

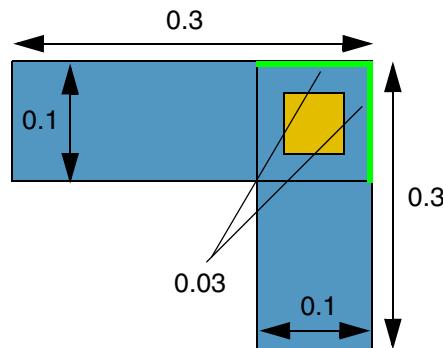
If joint width is less than 0.15 and span length is greater than 0.1, the extension of one of the joint edges (Metal1) must be greater than or equal to 0.05 or the extension of both joint edges must be greater than or equal to 0.03.

```
orderedSpacings(
  ( minViaJointExtension "Metal1" "Via1"
    'jointWidth 0.15
    'spanLength 0.1
    (0.05 0.03)
  )
); orderedSpacings
```

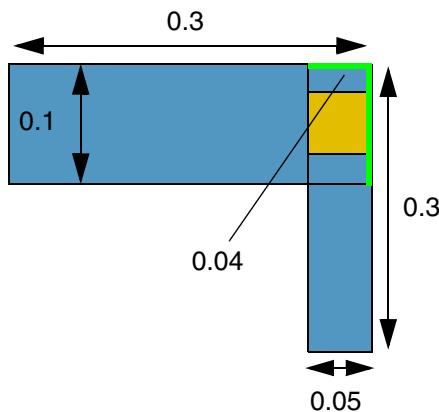
 Via1  
 Metal1



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 (ICADV12.3 Only)**

```
spacingTables(
  ( minVoltageExtension tx_layer1 tx_layer2
    ( "voltage" nil nil )
    g_table
  )
) ;spacingTables
```

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.

### **Values**

*tx\_layer1*                   The first layer on which the constraint is applied. The shape on this layer encloses.

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.

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

"voltage" nil nil

This identifies the index for *table*.

*g\_table*                   The format of the *table* row is as follows:

(*f\_voltage f\_extension*)

where, *f\_extension* is the extension of the *layer1* shape past the *layer2* shape. The *layer1* extension on all four sides must be greater than or equal to *f\_extension* if the voltage on *layer1* is greater than or equal to *f\_voltage*.

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

### **Parameters**

None

## Example

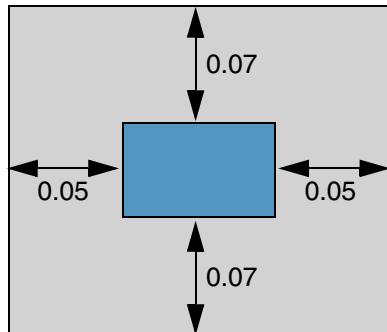
The extension of an Nwell shape past a Pdiff shape must be as follows:

- 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
```

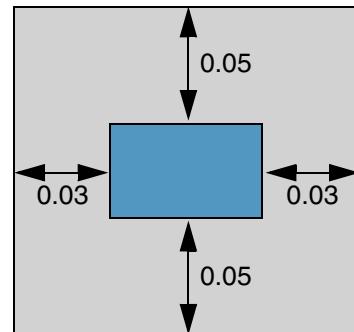
 Nwell  
 Pdiff

Nwell voltage:  $\geq 1.0$  to  $\leq 1.2$



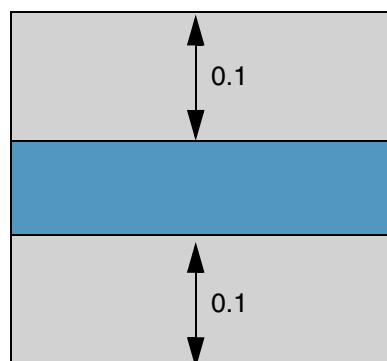
a) PASS. The voltage on Nwell is greater than or equal to 1.0 and the extensions on all sides are greater than or equal to 0.05.

Nwell voltage:  $\geq 1.0$  to  $\leq 1.2$



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:  $\geq 1.5$

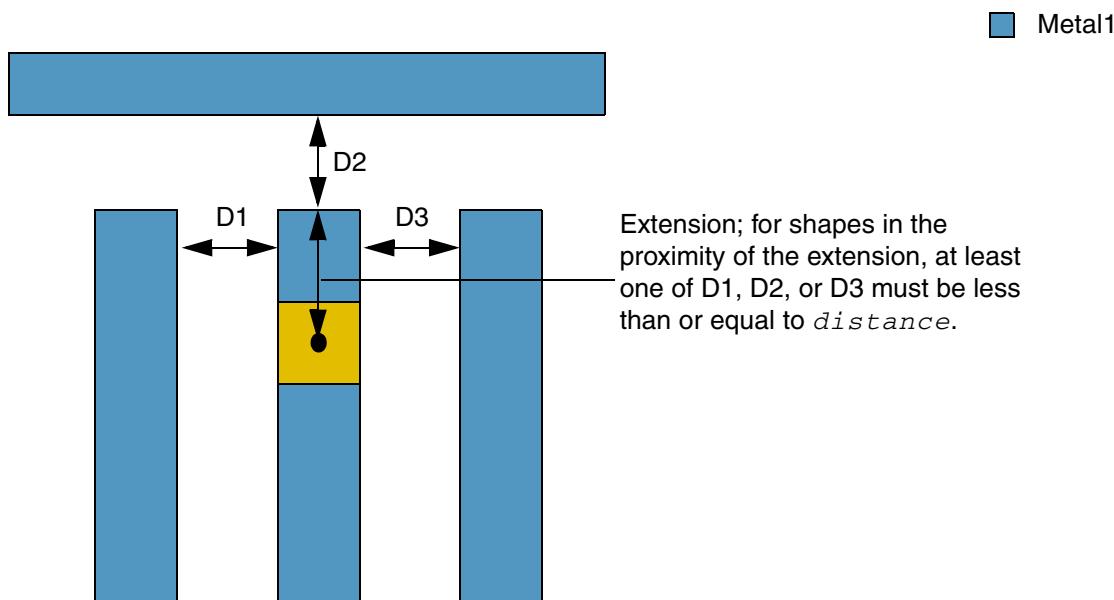


c) FAIL. The voltage on Nwell is greater than or equal to 1.5, but the extensions on the left and right sides are 0 ( $< 0.07$ ).

## minWireExtension

```
spacings(  
  ( minWireExtension tx_layer  
    ([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.

## 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 Constraint Reference**  
Extension Constraints

---

---

## Length and Width Constraints

---

This chapter includes the following constraints:

- [allowedGateLengthRanges](#)
- [allowedGateWidthRanges](#)
- [allowedLengthRanges \(Advanced Nodes Only\)](#)
- [allowedNeighborWidthRanges \(One layer\) \(Advanced Nodes Only\)](#)
- [allowedNeighborWidthRanges \(Two layers\) \(Advanced Nodes Only\)](#)
- [allowedNeighborWidthRangesOver \(Advanced Nodes Only\)](#)
- [allowedPRBoundaryDimensions \(Advanced Nodes Only\)](#)
- [allowedSpanLengthRanges](#)
- [allowedWidthRanges](#)
- [illegalHGatePattern](#)
- [maxDiagonalEdgeLength](#)
- [maxDiffusionLength](#)
- [maxLength](#)
- [maxNumMinEdges](#)
- [maxPolyLength](#)
- [maxTouchingDirectionLength](#)
- [maxWidth](#)
- [minChamferLength](#)
- [minDiagonalEdgeLength](#)
- [minDiagonalWidth](#)

## Virtuoso Technology Data Constraint Reference

### Length and Width Constraints

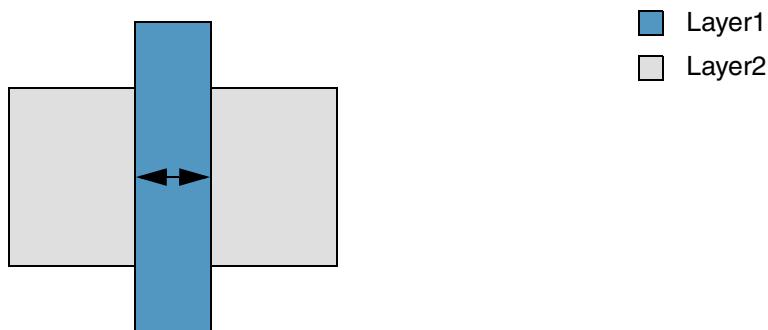
---

- [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 Constraint Reference

### Length and Width Constraints

---

#### Example

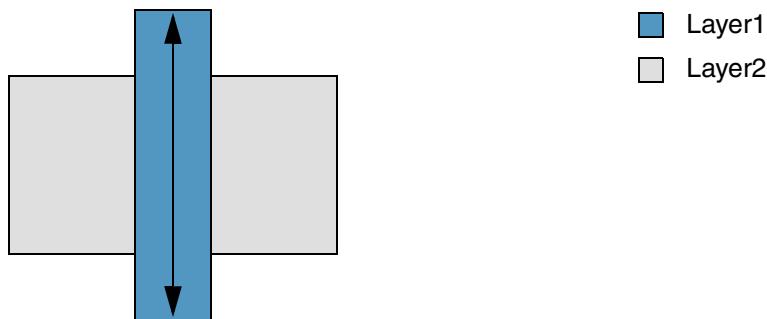
The channel length must be greater than 0.2 and less than 0.4 or greater than or equal to 1.0.

```
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 Constraint 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
```

## **allowedLengthRanges (Advanced Nodes Only)**

```

    spacings(
        ( allowedLengthRanges tx_layer
            (g_ranges)
        )
    ) ;spacings

spacingTables(
    ( allowedLengthRanges tx_layer
        (( "width" nil nil )
            [f_default]
        )
        (g_table)
    )
) ;spacingTables

```

Specifies the allowed length values for shapes on a layer.

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 indices are the widths specified with the allowedWidthRanges constraint.

## Values

*tx\_layer* The layer on which the constraint is applied.

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

*g\_ranges* The allowed length ranges.

Type: Floating-point values specifying length ranges.

"width" nil nil

This identifies the index for *table*.

*g\_table*

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

(*f\_width* (*g\_ranges*))

where,

- *f\_width* is the width of the shape.
- *g\_ranges* is a list of the allowed length ranges for the specified width.

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 allowed lengths for rectangles on Metal1 are determined as follows:

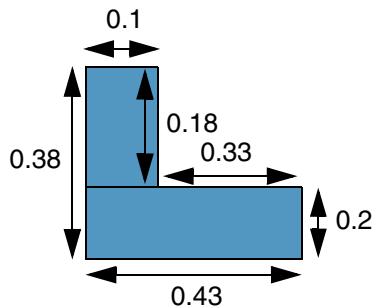
- If width is 0.01, the length must be greater than or equal to 0.1 and less than or equal to 0.18.
- If width is 0.02, the length must be greater than or equal to 0.3 and less than or equal to 0.43.
- If width is 0.04, the length must be greater than or equal to 0.3.

## Virtuoso Technology Data Constraint Reference

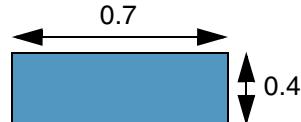
### Length and Width Constraints

```
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) (Advanced Nodes Only)**

```
spacingTables(
  ( allowedNeighborWidthRanges tx_layer
    (( "width" nil nil )
     'distanceWithin f_within
     ['widthRanges (g_widthRanges)]
     [f_default]
    )
    (g_table)
  )
) ;spacingTables
```

Specifies the width of a shape that is within a certain distance of a neighboring shape on the same layer.

### **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* (*g\_ranges*))

where,

- *f\_width* is the width of the neighboring shape.
- *g\_ranges* is a list of the allowed width ranges, 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.

## 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](#).

*f\_default*

The width value to be used when no table entry applies.

## 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 Constraint Reference

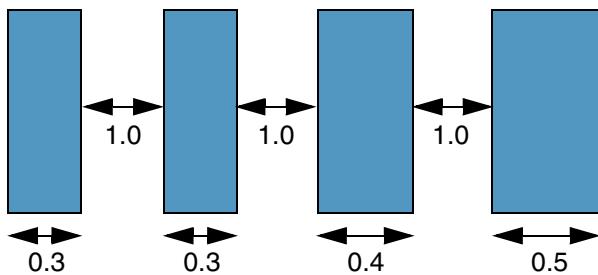
## Length and Width Constraints

```

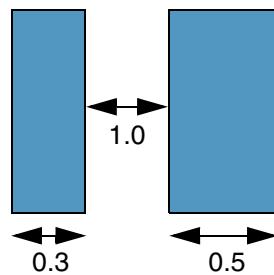
spacingTables(
    allowedNeighborWidthRanges "Metall1"
        (( "width nil nil"
            'distanceWithin 1.5
            'widthRanges (" [0 1.0] ")
        )
        (
            0.3 (0.3 0.4)
            0.4 (0.3 0.4 0.5)
            0.5 (0.4 0.5)
        )
    )
) ;spacingTables

```

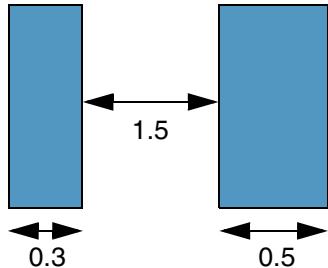
 Metal1



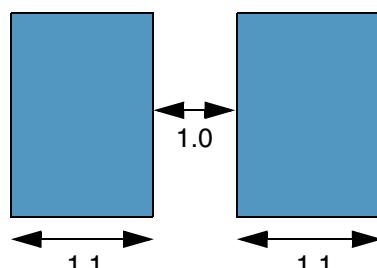
a) PASS. The width of all shapes is in the range [0 1.0] and the distance between each pair of shapes is 1.0 (<1.5). Each shape has neighboring shapes that satisfy the width requirement.



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) (Advanced Nodes Only)**

```
    spacingTables(
        ( allowedNeighborWidthRanges tx_layer1 tx_layer2
            (( "width" nil nil )
                'distanceWithin f_within
                [ 'widthRanges (g_widthRanges) ]
                [ 'otherWidthRanges (g_otherWidthRanges) ]
                [ f_default ]
            )
            (g_table)
        )
    )
); spacingTables
```

Specifies the width of a *layer1* shape that is within a certain distance of a neighboring *layer2* shape.

## Values

*tx\_layer1* The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

"width" nil nil

This identifies the index for *table*.

*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.

## 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.

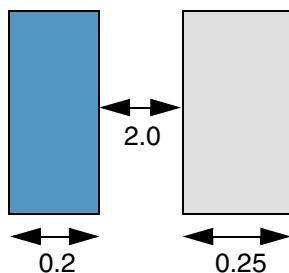
# Virtuoso Technology Data Constraint Reference

## Length and Width Constraints

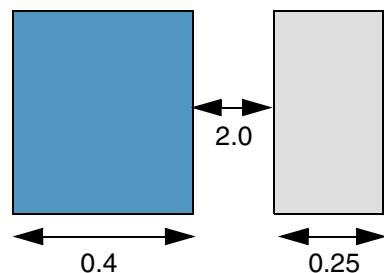
---

```
spacingTables(
  allowedNeighborWidthRanges "Metal1" "Metal1"
    ( ( "width nil nil" )
      'distanceWithin 2.5
    )
    (
      0.25 ( "[ 0.2 0.3 ]" )
    )
)
) ;spacingTables
```

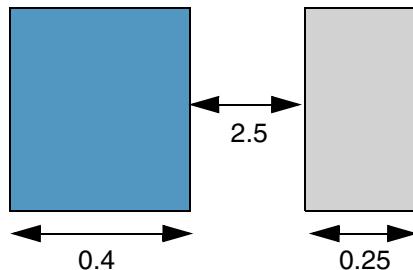
 Metal2  
 Metal1



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 (Advanced Nodes Only)

```
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
```

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> .

*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 Constraint Reference

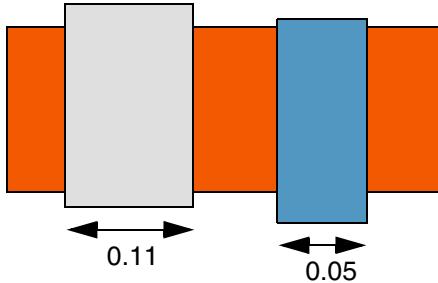
### Length and Width Constraints

#### Example

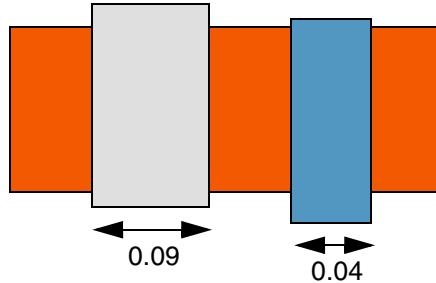
The width of a Metal1 shape must be 0.04, 0.05, or 0.06. If neighboring Metal1 and Metal2 shapes overlap a Metal3 shape and the width of a Metal2 shape is greater than or equal to 0.1, the width of the Metal1 shape must be 0.05 or 0.06.

```
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 0.9 ( $<0.1$ ). The width of the Metal1 shape is 0.04, which satisfies the allowedWidthRanges constraint.

#### **allowedPRBoundaryDimensions (Advanced Nodes Only)**

```

    spacings(
        ( allowedPRBoundaryDimensions
            [ 'horizontal' | 'vertical' ]
            [ 'stepSize' f_stepSize ]
            (g_ranges)
        )
    ) ;spacings

```

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 ( $D_{\text{legal}}$ ) are restricted to the following values, up to the upper bound of each range:

$$D_{legal} = D_{lb} + n^* stepSize$$

where,  $D_{lb}$  is the lower bound of each range and  $n >= 0$ .

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 Constraint Reference

### Length and Width Constraints

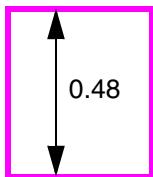
---

#### Example

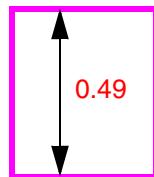
The vertical distance between the facing edges of a PR boundary must be greater than or equal to 0.24 and less than or equal to 4.8. A legal dimension value is the sum of the lower bound of this distance range (0.24) and a multiple of 0.048.

```
spacings(
  allowedPRBoundaryDimensions
    'vertical
    'stepSize 0.048
    (" [0.24 4.8] ")
)
) ; spacings
```

 PR boundary



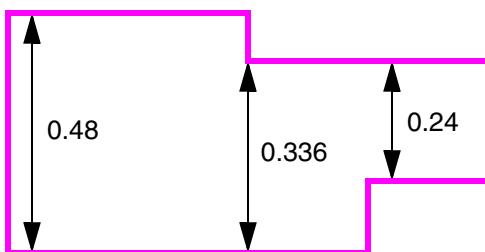
a) PASS. The vertical 0.48 distance (for n=5) is an allowed value that falls in the specified dimensions range.



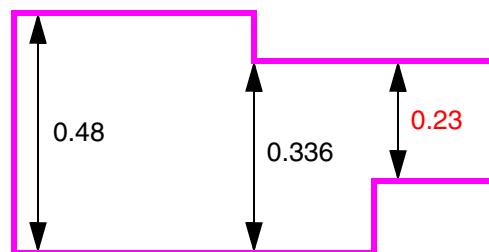
b) FAIL. The distance 0.49 is not a legal value.



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]
    [ 'orthogonalLength f_length]
    [ 'otherSpanLength f_spanLength]
    (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
'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>	(ICADV12.3 Only) The constraint applies only if the span length measured in the direction perpendicular to the specified direction is greater than this value.  <b>Note:</b> Specify this parameter only if the <code>rectShapeDir</code> constraint is specified for the layer.

## Examples

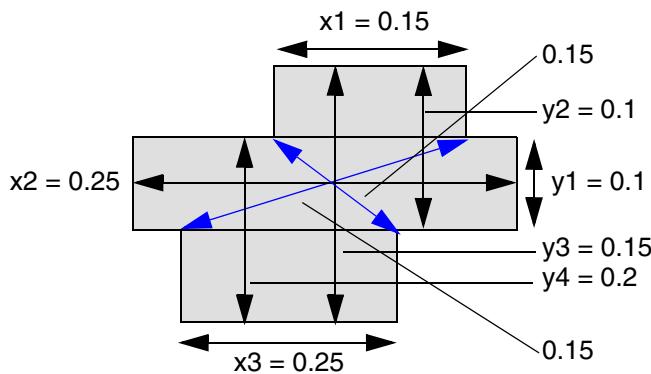
- [Example 1: allowedSpanLengthRanges with horizontal, vertical, and orthogonalLength](#)
- [Example 2: allowedSpanLengthRanges with horizontal, vertical, and otherSpanLength](#)

### ***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 equal to 0.1, 0.15, or 0.2, or is greater than 0.3 and less than 15, and the distance between two inside-facing corners is greater than or equal to 0.13.
- The span length measured horizontally is equal to 0.15 or 0.25, or is 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
```



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 ( $>0.13$ ), indicated by blue arrows. Additionally, the horizontal span lengths  $x_1$ ,  $x_2$ , and  $x_3$  are 0.15, 0.25, and 0.25, respectively.

## Virtuoso Technology Data Constraint Reference

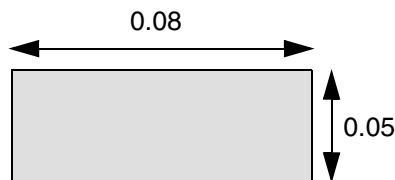
### Length and Width Constraints

#### **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 equal to 0.05, 0.06, 0.07, 0.08 or 0.09, or is greater than 0.1 and less than 15.
- The span length measured horizontally is equal to 0.07, or is 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(           □ Metal2
  ( 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
```



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.

## **allowedWidthRanges**

```
spacings(
  ( allowedWidthRanges tx_layer
    ['length f_length]
    ['orthogonal]
    ['horizontal | 'vertical | 'measureHorizontal | 'measureVertical]
    ['stepSize f_stepSize ['stepRange g_range]]
    (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.

### **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**

'length <i>f_length</i>	The constraint applies only if the length of the shape is greater than or equal this value.
'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

## Virtuoso Technology Data Constraint Reference

### Length and Width Constraints

---

'horizontal | 'vertical | 'measureHorizontal | 'measureVertical

The parameters 'horizontal and 'vertical, if specified, indicate the direction in which width is measured.

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*

(Advanced Nodes Only) The allowed widths ( $W_{\text{legal}}$ ) are restricted to the following values:

$$W_{\text{legal}} = W_{\text{lb}} + n * stepSize$$

where,  $W_{\text{lb}}$  is the lower bound of the range specified by *stepRange* and  $n \geq 0$ . If *stepRange* is not specified,  $W_{\text{lb}}$  is the lower bound of the constraint value.

'stepRange *g\_stepRange*

(Advanced Nodes Only) 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.

## 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](#)

## Virtuoso Technology Data Constraint Reference

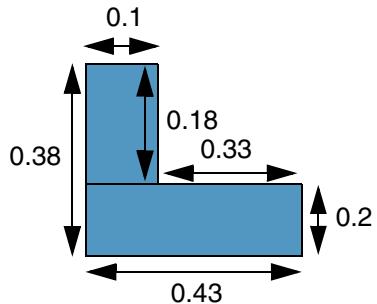
### Length and Width Constraints

#### **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).

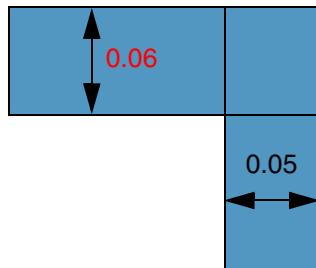
**Example 2: allowedWidthRanges with vertical and orthogonal**

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.

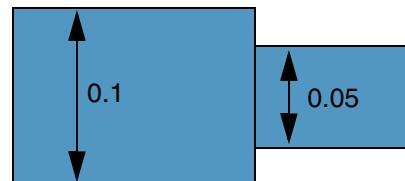
```
spacings(
  ( allowedWidthRanges "Metal1"
    'vertical
    'orthogonal
    (0.05 0.07 0.1 0.12 0.15)
  )
) ;spacings
```

 Metal1

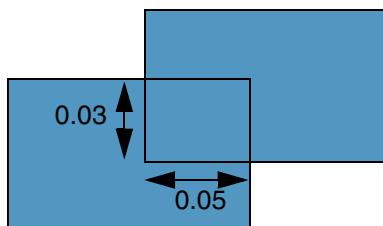
Preferred routing direction: Vertical



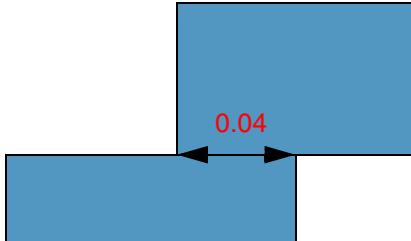
a) FAIL. The width 0.06 measured vertically is not an allowed width.



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 Constraint 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
```

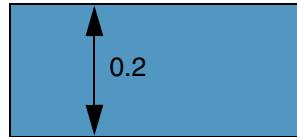
Metal1



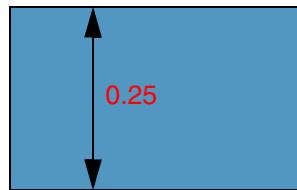
a) PASS. The constraint is satisfied for n=0.



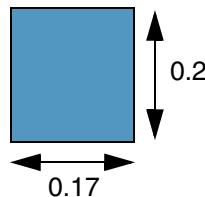
b) PASS. The constraint is satisfied for n=1.



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 Constraint Reference

### Length and Width Constraints

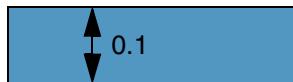
---

#### **Example 4: allowedWidthRanges with stepSize and stepRange**

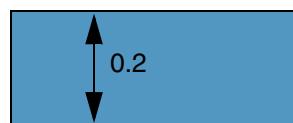
The vertical width of a wire must be greater than or equal to 0.1 and less than or equal to 0.4. A legal width value is also the sum of the lower bound of this width range (0.1) and a multiple of 0.05. For widths greater than or equal to 0.1 and less than or equal to 0.21, only discrete widths at a 0.05 step are allowed.

```
spacings(
  ( allowedWidthRanges "Metall1"
    'stepSize 0.05
    'stepRange [0.1 0.21]
    ("[0.1 0.4]")
  )
) ; spacings
```

 Metal1



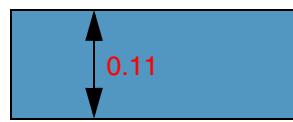
a) PASS. The constraint is satisfied for n=0.



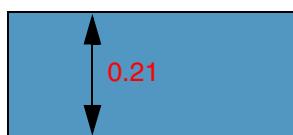
b) PASS. The constraint is satisfied for n=2.



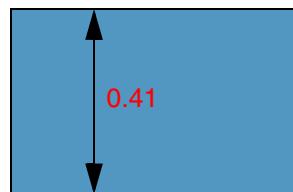
c) PASS. The constraint is satisfied for n=4.



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.



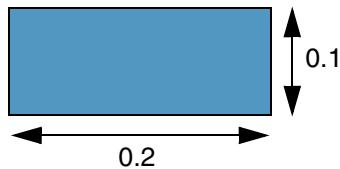
f) FAIL. The maximum allowed width is 0.41.

**Example 5: allowedWidthRanges with measureVertical and measureHorizontal**

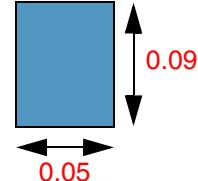
Two allowedWidthRanges constraints are defined in an AND constraint group. A check is performed in the specified direction, regardless of whether this direction is the width or length of the shape.

```
spacings(
  ( allowedWidthRanges "Metal1"
    'measureVertical
    (">=0.1")
  )
  ( allowedWidthRanges "Metal1"
    'measureHorizontal
    (">=0.2")
  )
) ;spacings
```

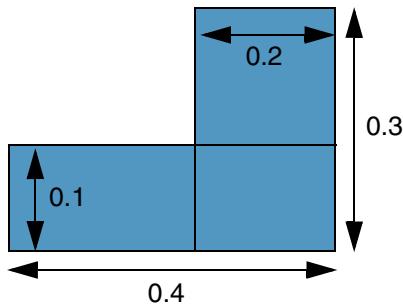
Metal1



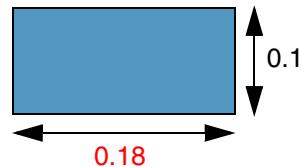
a) PASS. The constraint is satisfied in both directions.



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 Constraint Reference

### Length and Width Constraints

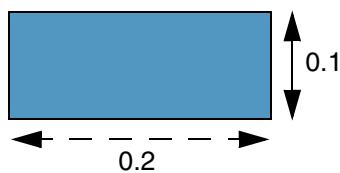
---

#### ***Example 6: allowedWidthRanges with vertical and horizontal***

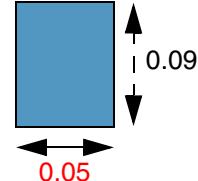
Two allowedWidthRanges constraints are defined in an AND constraint group. A check is performed in the specified direction for the shape width, which is the shorter of the two dimensions. A check is not performed on the length represented by the dashed arrows.

```
spacings(
  ( allowedWidthRanges "Metall1"
    'vertical
    (">=0.1")
  )
  ( allowedWidthRanges "Metall1"
    'horizontal
    (">=0.2")
  )
) ;spacings
```

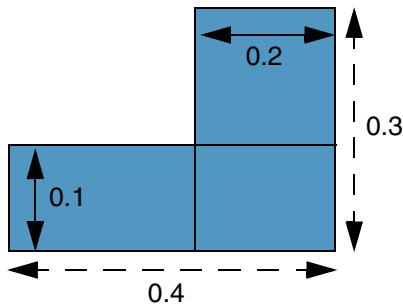
 Metal1



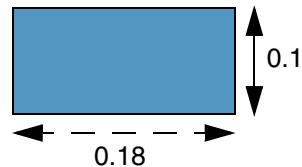
a) PASS. The constraint is satisfied in the vertical direction.



b) FAIL. The constraint is violated in the horizontal direction.



c) PASS. The constraint is satisfied in both directions for each rectangle.



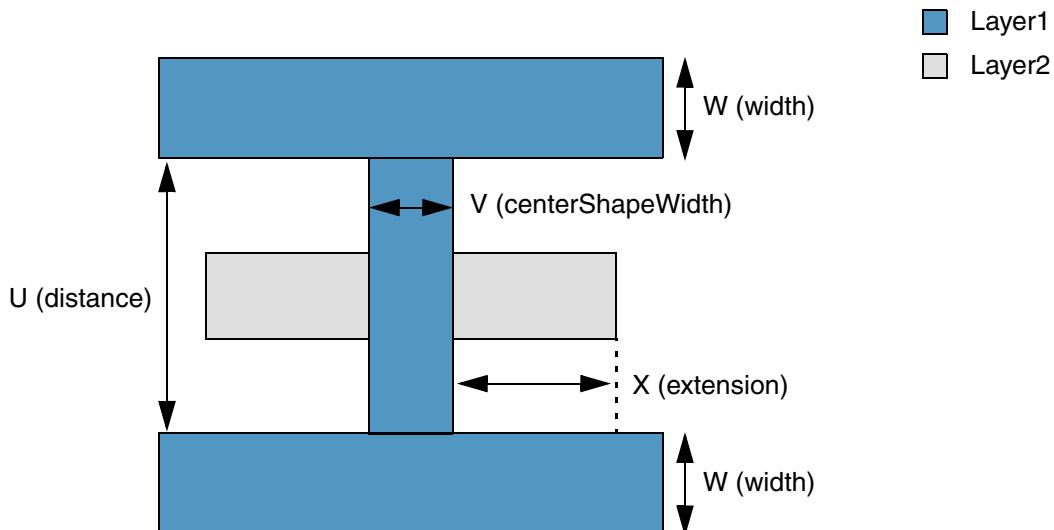
d) PASS. The constraint is satisfied in the vertical direction.

## 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 Constraint 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

'width <i>f_width</i>	The width of the rails must be greater than or equal to this value.
'length <i>f_length</i>	The extension of the rails from the connecting shape must be less than or equal to this value.
'within <i>f_distance</i>	The distance between the rails must be greater than or equal to this value.

#### Example

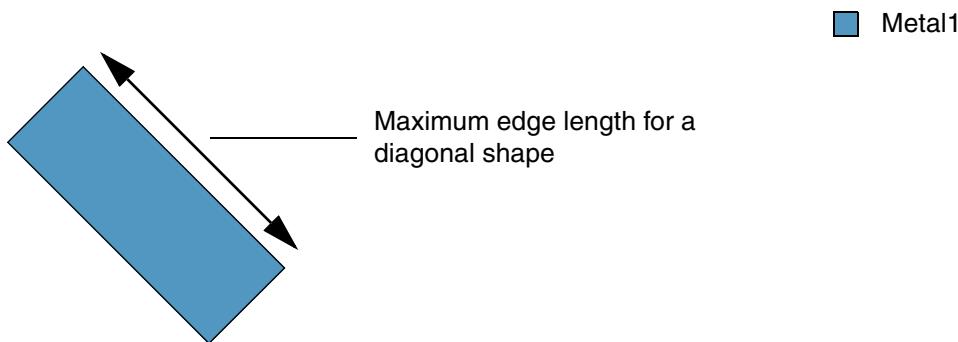
If shapes are connected to form an H-shaped pattern on Metal1, the width of the rails must be greater than or equal to 0.3, the extension of the rails from the connecting shape must be less than or equal to 0.9, the distance between the rails must be greater than or equal to 0.5, and the width of the 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. The distance should be calculated as the actual or Euclidian diagonal distance.



### **Values**

*tx\_layer*                          The layer on which the constraint is applied.

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

*g\_length*                          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
```

## **maxDiffusionLength**

```
orderedSpacings(
  ( maxDiffusionLength tx_layer1 tx_layer2
    [ 'width f_width
      f_length
    ]
  )
) ;orderedSpacings
```

Specifies the maximum length of the diffusion layer between two cut shapes or between a cut shape and the line-end of the diffusion layer.

### **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> or between an edge of a <i>layer2</i> shape and the nearest <i>layer1</i> line-end must be less than this value.

### **Parameters**

'width <i>f_width</i>	The constraint applies only if the width of the <i>layer1</i> shape is less than this value.
-----------------------	--

## Virtuoso Technology Data Constraint Reference

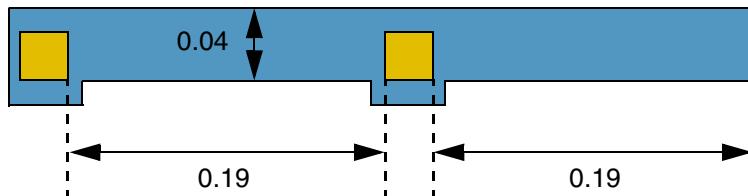
### Length and Width Constraints

#### Example

The distance between the two Via1 edges and a Via1 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" "Via1"  
        'width 0.05  
        0.2  
    )  
) ;orderedSpacings
```

 Via1  
 Diffusion

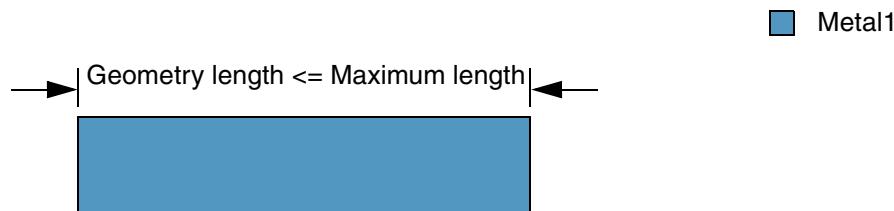


PASS. The width of the shape on Diffusion layer is 0.04 (<0.05) and the distance between the two Via1 edges and the Via1 edge and the Diffusion line-end is 0.19 (<0.2).

## **maxLength**

```
spacings(  
  ( maxLength tx_layer 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**

None

### **Example**

The maximum length of a shape on Metal1 must be 7.0.

```
spacings(  
  ( maxLength "Metal1" 7.0 )  
) ;spacings
```

## **maxNumMinEdges**

```
spacings(
  ( maxNumMinEdges tx_layer (f_length x_numEdges) )
) ;spacings
```

Sets the maximum number of consecutive edges that are shorter than the specified 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_length</i>	The length of an edge that is counted must be less than this value.
<i>x_numEdges</i>	The number of edges shorter than <i>length</i> must be less than or equal to this value.

### **Parameters**

None

## Virtuoso Technology Data Constraint Reference

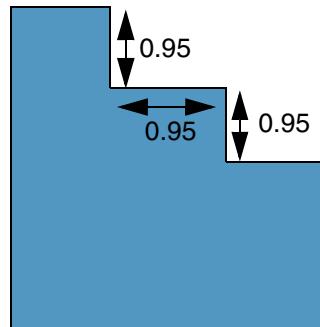
### Length and Width Constraints

#### Example

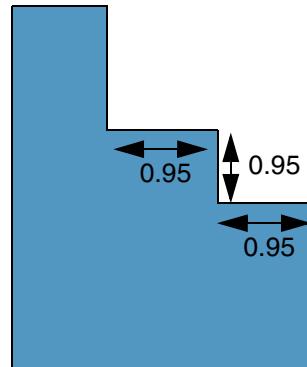
The maximum number of consecutive edges shorter than 1.0 must be 2.

```
spacings(  
    ( maxNumMinEdges "Metal1" (1.0 2) )  
) ;spacings
```

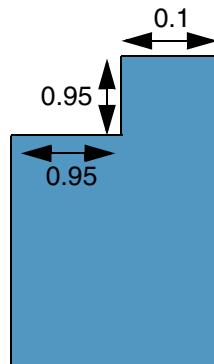
 Metal1



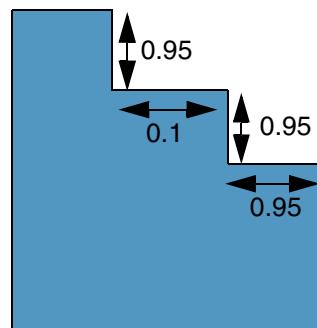
a) FAIL. The number of consecutive edges shorter than 1.0 is 3 (>2).



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.

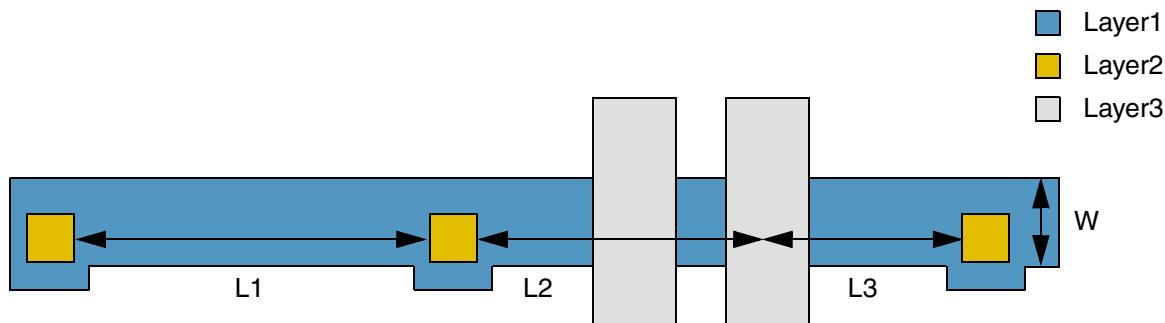


d) PASS. The number of consecutive edges shorter than 1.0 is 2.

## 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> .

## Parameters

'width *f\_width*

The constraint applies only if the width of the *layer1* shape is less than this value.

## Example

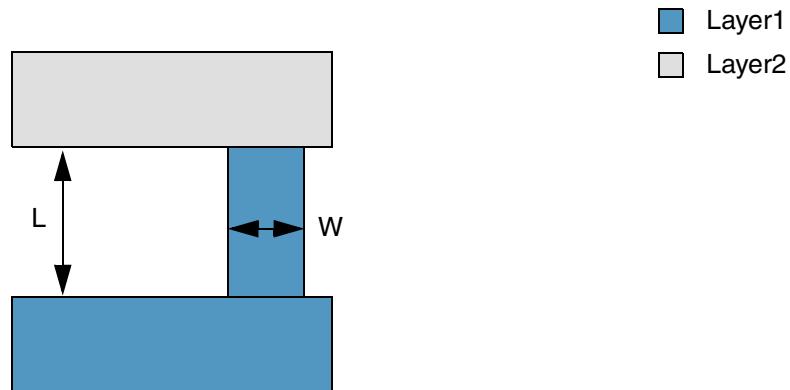
The maximum allowed distance between Cont edges, along Poly, or from any point on Oxide over Poly, along Poly, must be 0.5 when the width of the Poly shape is less than 0.2.

```
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> .

*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
```

## **maxWidth**

```
spacings(  
  ( maxWidth tx_layer f_width )  
) ;spacings
```

Specifies the maximum width 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_width</i>	The width of the shape must be less than or equal to this value.

## **Parameters**

None

## **Example**

The maximum width of shapes on Metal3 must be 2.5.

```
spacings(  
  ( maxWidth "Metal3" 2.5 )  
) ;spacings
```

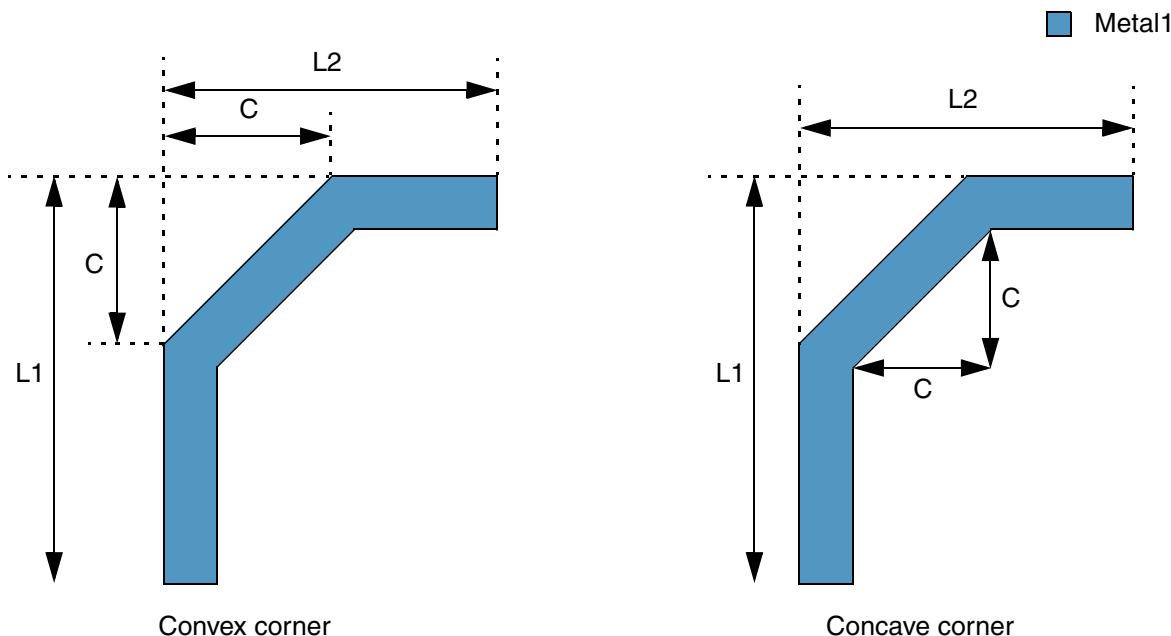
## minChamferLength

```
spacingTables(  
    ( minChamferLength tx_layer  
        (( "length" nil nil )  
         [ 'convex | 'concave ]  
        )  
        g_table  
    )  
) ; spacingTables
```

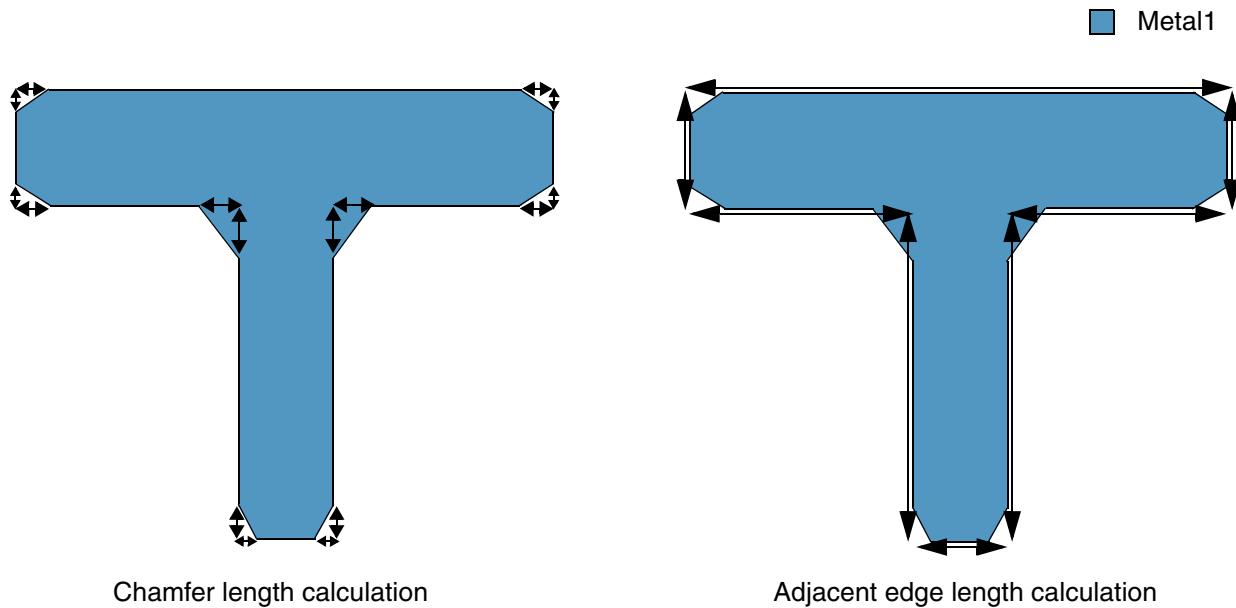
Specifies required length of chamfer, indexed by adjacent edge lengths.

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.



## 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.

## Examples

- [Example 1: minChamferLength with one corner](#)
- [Example 2: minChamferLength with multiple corners](#)

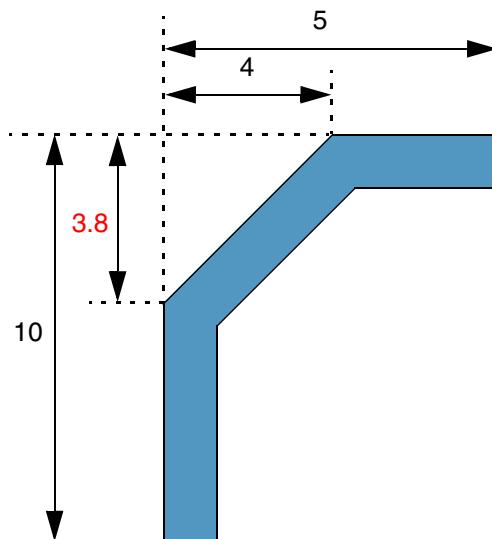
### ***Example 1: minChamferLength with one corner***

The chamfer length for concave and convex corners must be at least:

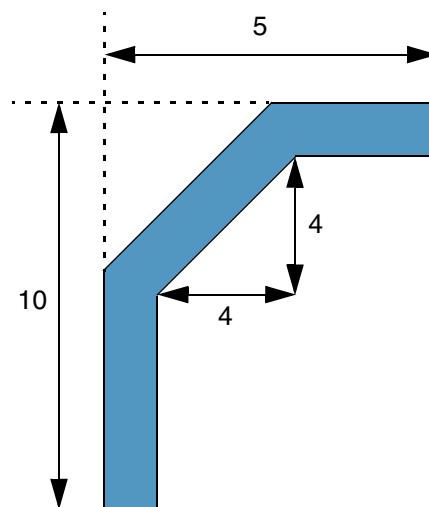
- 3.0 when adjacent edge length is 5
- 4.0 when adjacent edge length is 10
- 5.0 when adjacent edge length is 15

```
spacingTables(
    minChamferLength "Metal1"
        (( "length" nil nil ))
    (
        5      3.0
        10     4.0
        15     5.0
    )
)
) ;spacingTables
```

█ Metal1



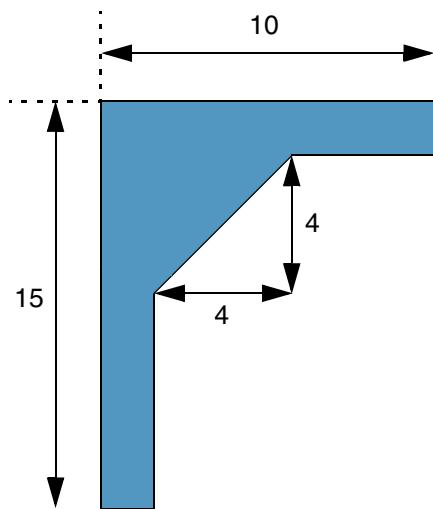
a) FAIL. When adjacent edge width is 10, the chamfer length must be at least 4.



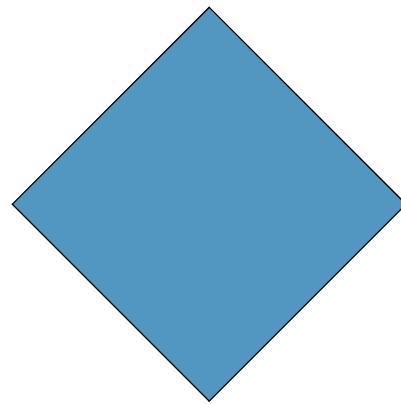
b) PASS. Chamfer length is satisfied for adjacent edge widths 5 and 10.

## Virtuoso Technology Data Constraint 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.

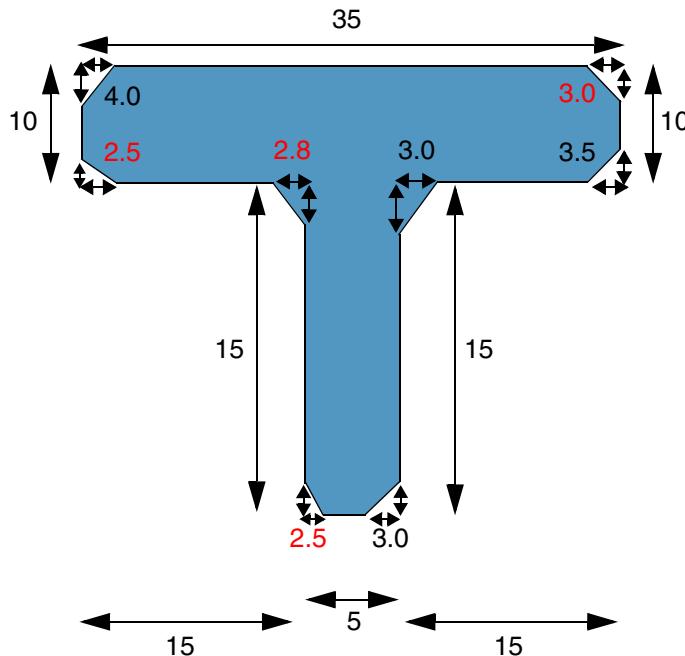
**Example 2: minChamferLength with multiple corners**

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

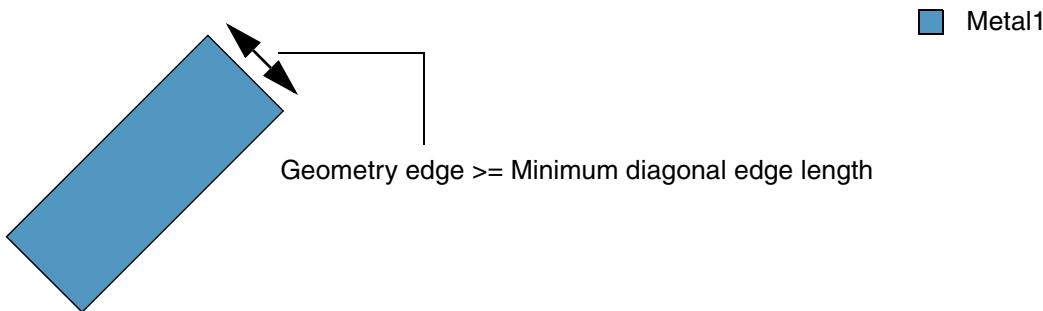


FAIL. Top two corners need a chamfer length of at least 4.0 because the adjacent edge length is 35, while the remaining corners need a chamfer length of at least 3.0 because the maximum adjacent edge length is 15.

## **minDiagonalEdgeLength**

```
spacings(  
  ( minDiagonalEdgeLength tx_layer f_length )  
) ;spacings
```

Sets the minimum edge length for a diagonal shape on the specified layer. The distance should be calculated as the actual or Euclidian diagonal distance.



### **Values**

*tx\_layer*      The layer on which the constraint is applied.

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

*f\_length*      The minimum length of a diagonal edge.

### **Parameters**

None

### **Example**

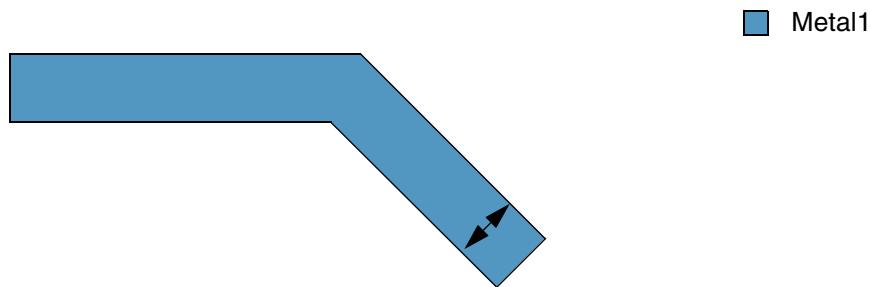
The minimum diagonal edge length must be 1.5 on Metal1 and equal to the value of the technology parameter `diaglength1` on Metal2.

```
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**

*tx\_layer*      The layer on which the constraint is applied.

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

*g\_width*      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 `mindiagwidth1` on Metal2.

```
spacings(  
  ( minDiagonalWidth "Metal1" 1.0 )  
  ( minDiagonalWidth "Metal2" techParam("mindiagwidth1") )  
) ;spacings
```

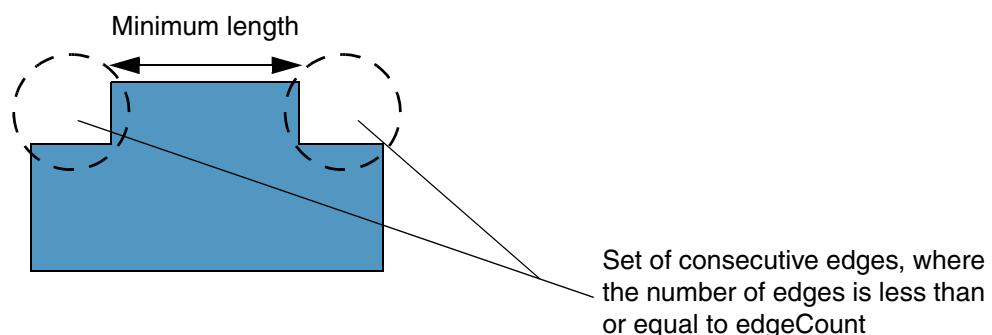
## **minEdgeAdjacentDistance**

```
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 Constraint 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

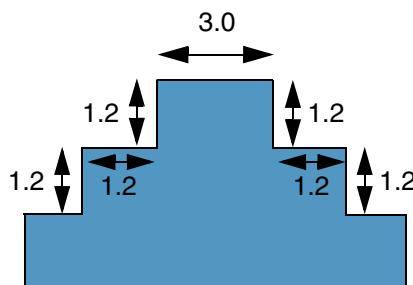
'length <i>f_length</i>	Each connected edge must have length less than this value.
'edgeCount <i>x_edgeCount</i>	The number of connected edges shorter than <i>length</i> must be less than or equal to this value.
'exceptSameCorner	The constraint does not apply to an edge that has the same type of 90-degree 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.

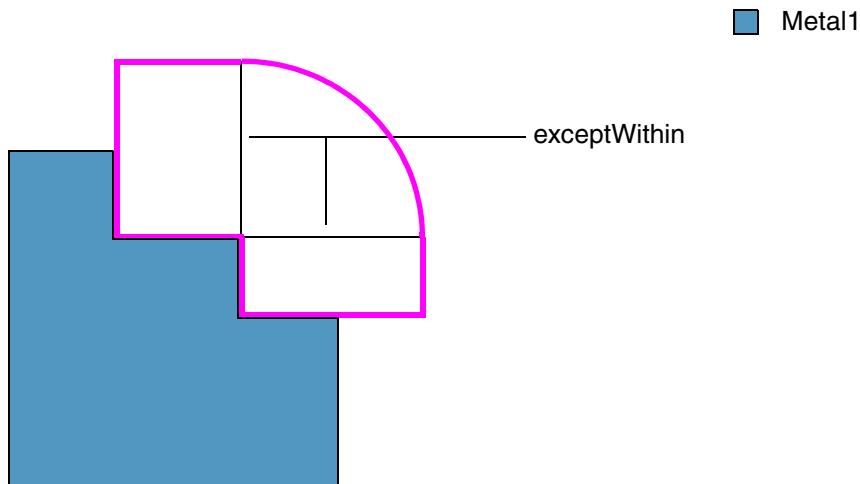
## **minEdgeAdjacentLength**

```
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 Constraint Reference

### Length and Width 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>f_length</code>	The lengths of both adjacent edges must be greater than or equal to this value.
<code>f_length1 f_length2</code>	The length of one adjacent edge must be greater than or equal to <code>length1</code> and the length of the other adjacent edge must be greater than or equal to <code>length2</code> . The two lengths can be identical.

#### Parameters

<code>'maxLength f_maxLength</code>	The constraint applies only if the length of a short edge is less than this value.
<code>'edgeCount x_maxEdges</code>	The constraint applies only if the number of connected edges with length less than <code>maxLength</code> is less than or equal to this value.
<code>'convexCorner</code>	If one edge of a convex corner that lies between two concave corners is less than <code>length</code> (the constraint value), the other edge must be greater than or equal to <code>maxLength</code> . The constraint does not apply if a convex corner is not found between two concave corners.
<code>'exceptWithin f_within</code>	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.
<code>'concaveCorner</code>	(Advanced Nodes Only) If one edge of a concave corner that lies between two convex corners is less than <code>maxLength</code> , the other edge must be greater than or equal to <code>length</code> (the constraint value). The constraint does not apply if a concave corner is not found between two convex corners.

'threeConcaveCorners

(ICADV12.3 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 *length* (the constraint value).

Type: Boolean

'width *f\_width*

(ICADV12.3 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 Constraint Reference

### Length and Width Constraints

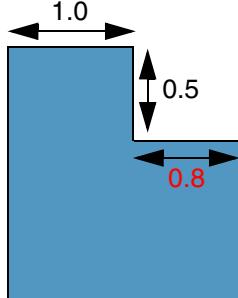
---

#### **Example 1: minEdgeAdjacentLength with maxLength and edgeCount**

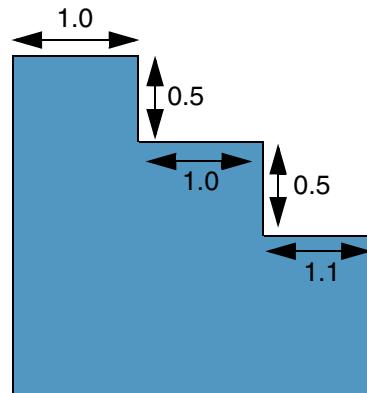
If a polygon has at most one short edge less than 0.6 long, the edges adjacent to the short edge must be at least 1.0 long.

```
spacings(
  ( minEdgeAdjacentLength "Metal1"
    'maxLength 0.6
    'edgeCount 1
    1.0
  )
) ;spacings
```

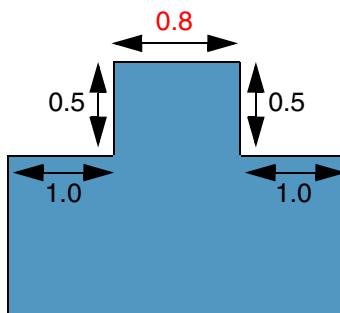
 Metal1



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 Constraint Reference

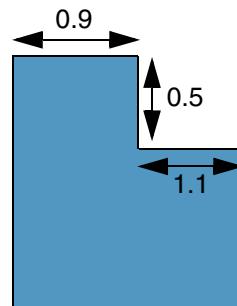
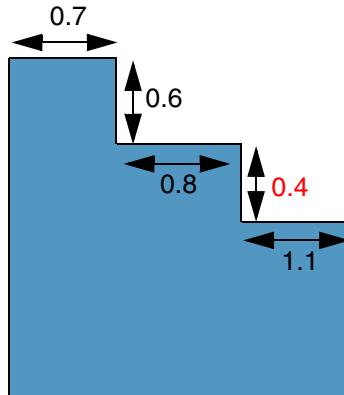
### Length and Width Constraints

#### **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.

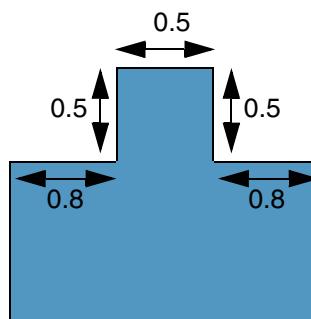
```
spacings(
  ( minEdgeAdjacentLength "Metal1"
    'maxLength 0.6
    'edgeCount 1
    'convexCorner
    1.0
  )
) ;spacings
```

 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.

**Example 3: minEdgeAdjacentLength with maxLength, edgeCount, threeConcaveCorners, and width**

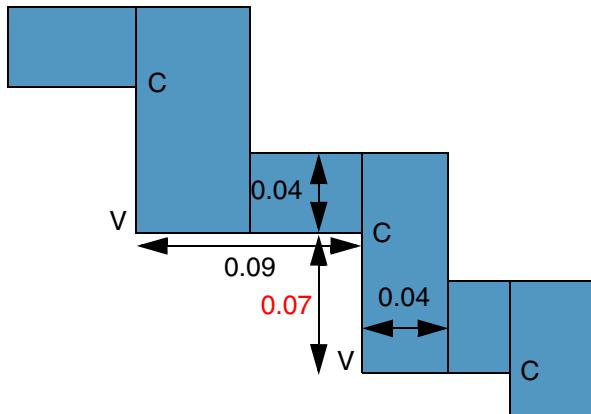
If there exist three consecutive concave (C) corners with two convex (V) corners in between, then the length of both edges that form the middle concave corner must be greater than or equal to 0.08, provided the shapes that are connected to form the middle concave corner are less than or equal to 0.04 wide.

```

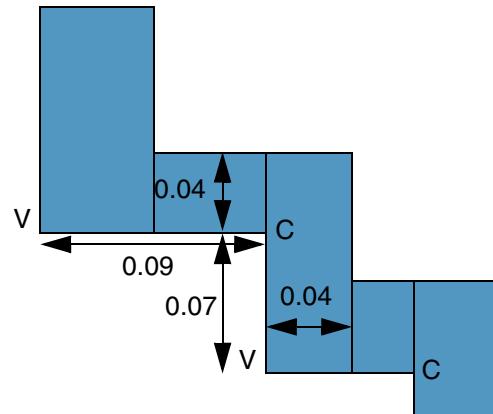
spacings(
  ( minEdgeAdjacentLength "Metal1"
    'maxLength 0.1
    'edgeCount 1
    'threeConcaveCorners
      'width 0.04
      0.08
  )
) ;spacings

```

■ Metal1



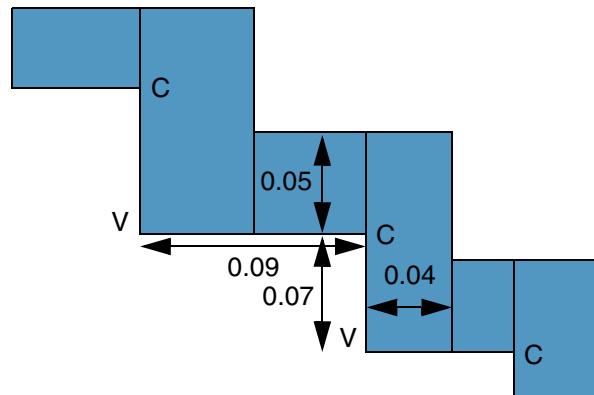
a) FAIL. The shapes that form the middle concave corner are 0.04 wide, but the length of the vertical edge that forms the middle concave corner is only 0.07 (<0.08).



b) The constraint does not apply because there exist only two concave corners.

## Virtuoso Technology Data Constraint 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**

*tx\_layer*                   The layer on which the constraint is applied.

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

*f\_length*                   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*

(Advanced Nodes Only) 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 Constraint 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	(Advanced Nodes Only) 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

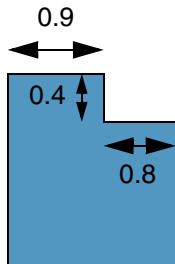
- [Example 1: minEndOfLineAdjacentToStep with maxLength and adjacentLength](#)
- [Example 2: minEndOfLineAdjacentToStep with maxLength and exceptAdjacentLength](#)
- [Example 3: minEndOfLineAdjacentToStep with maxLength and noBetweenEol](#)
- [Example 4: minEndOfLineAdjacentToStep with maxLength and concaveCorner](#)

**Example 1: *minEndOfLineAdjacentToStep* with *maxLength* and *adjacentLength***

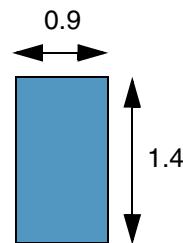
The length of an end-of-line edge must be greater than or equal to 1.0 if it is adjacent to an edge with length less than 1.5, which, in turn, forms a concave corner with an edge with length greater than or equal to 0.7 at the other end.

```
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 Constraint Reference

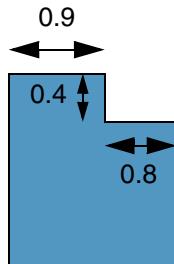
### Length and Width Constraints

#### **Example 2: minEndOfLineAdjacentToStep with maxLength and exceptAdjacentLength**

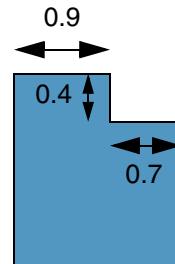
The length of an end-of-line edge must be greater than or equal to 1.0 if it is adjacent to an edge with length less than 1.5. The constraint does not apply if the minimum step edge forms a concave corner with an edge with length greater than or equal to 0.8 at the other end.

```
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).



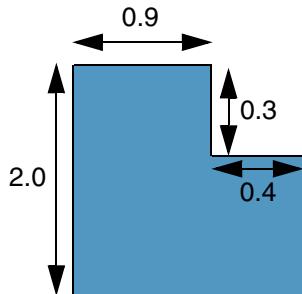
- b) FAIL. One edge adjacent to the edge with length 0.4 (<1.5) is an end-of-line edge and the length of the other adjacent edge is 0.7 (<0.8), but the length of the end-of-line edge is only 0.9 (<1.0).

**Example 3: *minEndOfLineAdjacentToStep* with *maxLength* and *noBetweenEol***

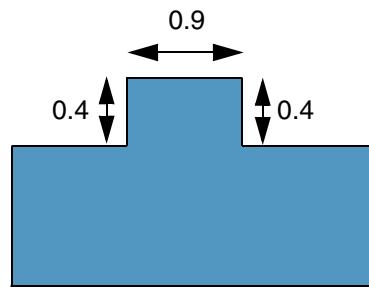
The length of an end-of-line edge must be greater than or equal to 1.0 if has two adjacent edges with length less than 1.5.

```
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.



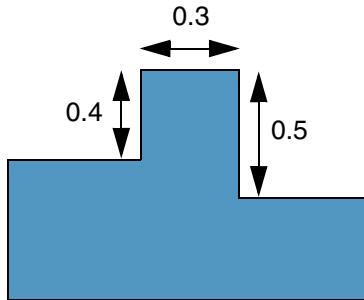
- b) FAIL. The end-of-line edge has two adjacent edges with length 0.4 (<1.5), but the length of the end-of-line edge is only 0.9 (<1.0).

**Example 4: *minEndOfLineAdjacentToStep* with *maxLength* and *concaveCorner***

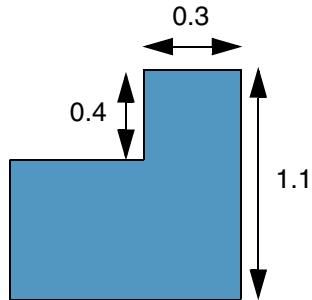
The length of an end-of-line edge must be greater than or equal to 1.0 if it has adjacent edges with length less than 1.5 and these adjacent edges form concave corners at their ends.

```
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).



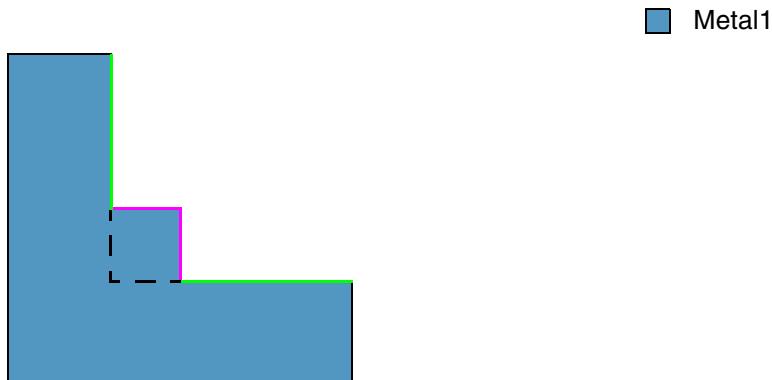
b) The constraint does not apply because the adjacent edge with length 1.1 (<1.5) does not form a concave corner at its other end.

## **minInsideCornerEdgeLength**

```
spacings(
  ( minInsideCornerEdgeLength tx_layer g_lengthDefinition )
) ;spacings
```

Sets the minimum length of each edge that is in a concave corner of a shape. A violation occurs if two or more consecutive edges of a concave corner are less than *edgeLength* long. If *lengthSum* is defined, a violation occurs only if the sum of lengths of all consecutive edges that are less than *edgeLength* long is greater than *lengthSum*.

The figure below illustrates a concave shape. For the constraint to be satisfied, the length of the two green edges must be is greater than or equal to *edgeLength* and the sum of the consecutive edges (in pink) must be less than or equal to *lengthSum*.



## Values

*tx\_layer*      The layer on which the constraint is applied.

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

*g\_lengthDefinition*

It takes one of the following forms:

- *f\_edgeLength*

The minimum step size or the shortest edge length. The edges must be greater than or equal to this value.

- *(f\_lengthSum f\_edgeLength)*

The sum of lengths of the consecutive edges (shown above in pink) that are shorter than *edgeLength* must be less than or equal to *lengthSum*.

## Parameters

None

## Example

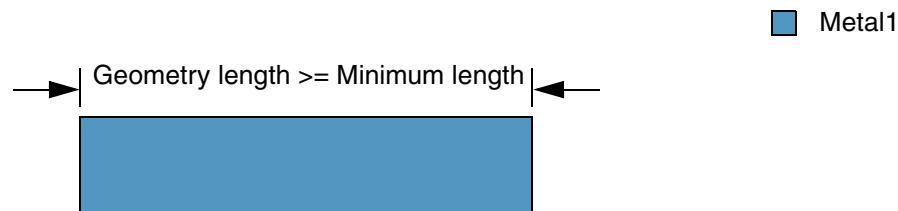
The minimum inside corner edge length on Poly1 and Metal2 must be 1.5. Additionally, on Metal2, the sum of lengths of the consecutive edges that are shorter than 1.5 must be less than or equal to 5.0.

```
spacings(
  ( minInsideCornerEdgeLength "Poly1" 1.5)
  ( minInsideCornerEdgeLength "Metal2" (5.0 1.5))
) ;spacings
```

## **minLength**

```
spacings(  
  ( minLength tx_layer 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**

None

### **Example**

The minimum orthogonal length for a shape on Metal1 must be 0.3.

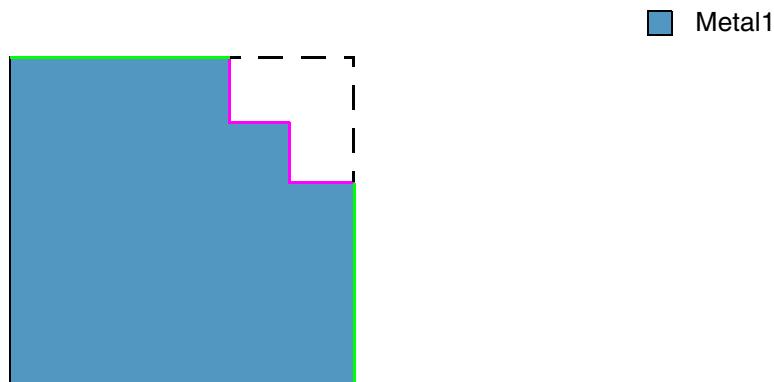
```
spacings(  
  ( minLength gate 0.3 )  
) ;spacings
```

## **minOutsideCornerEdgeLength**

```
spacings(  
  ( minOutsideCornerEdgeLength tx_layer g_lengthDefinition )  
) ;spacings
```

Sets the minimum length of each edge that is in a convex corner of a shape. A violation occurs if two or more consecutive edges of a convex corner are less than *edgeLength* long. If *lengthSum* is defined, a violation occurs only if the sum of lengths of all consecutive edges that are less than *edgeLength* long is greater than *lengthSum*.

The figure below illustrates a convex shape. For the constraint to be satisfied, the length of the two green edges must be greater than or equal to *edgeLength* and the sum of the consecutive edges (in pink) must be less than or equal to *lengthSum*.



## Values

*tx\_layer*      The layer on which the constraint is applied.

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

*g\_lengthDefinition*

It takes one of the following forms:

- *f\_edgeLength*

The minimum step size or the shortest edge length. The edges must be greater than or equal to this value.

- *(f\_lengthSum f\_edgeLength)*

The sum of lengths of the consecutive edges (shown above in pink) that are shorter than *edgeLength* must be less than or equal to *lengthSum*.

## Parameters

None

## Example

The minimum outside corner edge length on Poly1 and Metal2 must be 1.5. Additionally, on Metal2, the sum of lengths of the consecutive edges that are shorter than 1.5 must be less than or equal to 7.0.

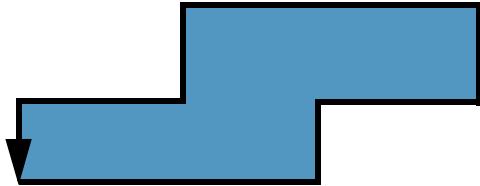
```
spacings(  
  ( minOutsideCornerEdgeLength "Poly1" 1.5)  
  ( minOutsideCornerEdgeLength "Metal2" (7.0 1.5))  
) ;spacings
```

## **minPerimeter**

```
spacings(  
  ( minPerimeter tx_layer f_perimeter )  
) ;spacings
```

Specifies the minimum perimeter of a shape on the given layer.

 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_perimeter</i>	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

'widthLengthTable (f_wideWidth f_length)	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> .
'enclosedWidth f_enclWidth	(Advanced Nodes Only) The width of the protrusion, excluding any portion of the wide wire, must be greater than or equal to this value.
'exceptCutTable (l_distance)	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.

```
'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

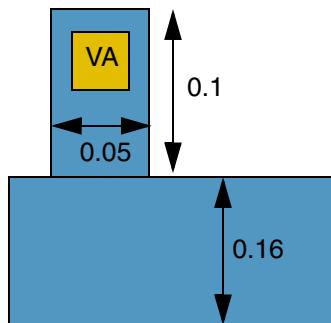
- [Example 1: minProtrusionWidthLength with widthLengthTable, exceptCutTable, and layer](#)
- [Example 2: minProtrusionWidthLength with widthLengthTable and enclosedWidth](#)

### **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.

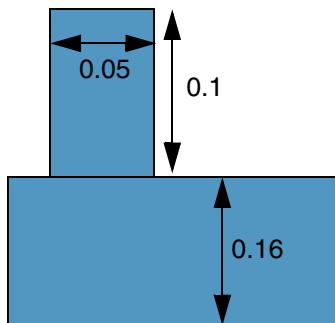
**Example 2: minProtrusionWidthLength with widthLengthTable and enclosedWidth**

The following conditions must be met:

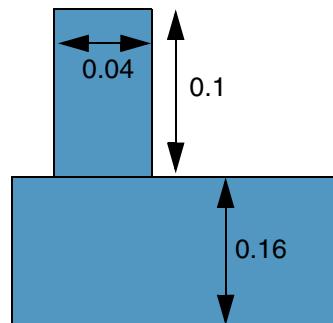
- The width of a protrusion on a wide Metal1 wire must be at least 0.05.
- If the width of the wide wire is greater than or equal to 0.11, the length of the protrusion must be at least 0.12.
- The width of the protrusion after excluding any width of the wide wire must be equal to 0.02.

```
spacings(
  ( minProtrusionWidthLength "Metal1"
    'widthLengthTable ( 0.11 0.12 )
    'enclosedWidth 0.02
    0.05
  )
) ;spacings
```

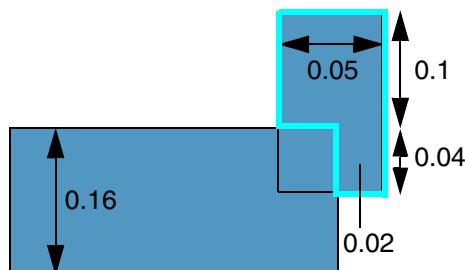
 Metal1



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.



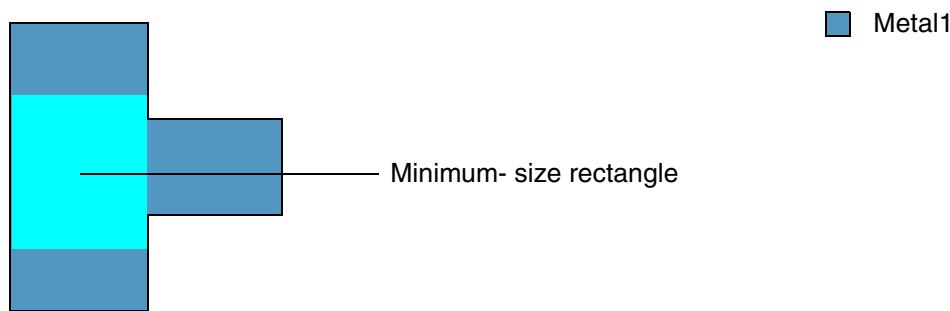
c) PASS. The width of the protrusion is 0.05 and its length 0.14 (>0.12). Its width excluding the width of the wide wire is 0.02.

## 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

<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 f_length</i> ) ...)	The width and length of the rectangle. The constraint value should be a list of lists, but a simple list can also be specified.

## Parameters

'rectOnly                          The constraint applies only to rectangular shapes.

## Examples

- [Example 1: minSize](#)
- [Example 2: minSize with rectOnly](#)

### ***Example 1: minSize***

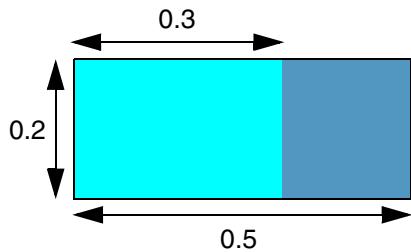
A 0.1x0.2 rectangle must fit inside every rectangular shape on Metal1.

```
spacings(
  ( minSize "Metal1"
    'rectOnly
    (0.1 0.2)
  )
) ;spacings
```

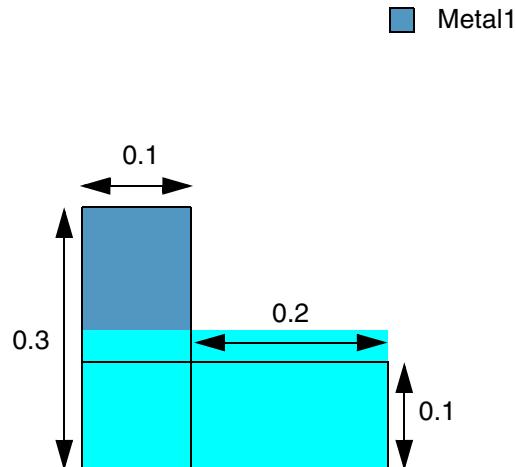
### ***Example 2: minSize with rectOnly***

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
```



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.

## minStepEdgeLength

```
spacings(
  ( minStepEdgeLength tx_layer
    ['any | 'horizontalEdge | 'verticalEdge]
    ['allCorner | 'concaveCorner | 'convexCorner | 'mixedCorner]
    ['exceptExactLength (f_exactLength f_adjLength) ...]
    ['exceptExactBothLength (f_exactLength f_exactAdjLength) ...]
    g_lengthSpec
  )
) ;spacings
```

Specifies the minimum edge length for a shape on the specified layer. The length of each such edge must be greater than or equal to *length* or the sum of the lengths of the edges must be less than or equal to *lengthSum*.

### Values

*tx\_layer*      The layer on which the constraint is applied.  
Type: String (layer and purpose names) or Integer (layer number)

*g\_lengthSpec*      The minimum edge length is specified using one of the following formats:

*f\_length* or (*f\_lengthSum f\_length*)

where:

- *f\_length* is the minimum length of each edge.
- *f\_lengthSum* is the maximum sum of the lengths of the consecutive edges.

### Parameters

'any | 'horizontalEdge | 'verticalEdge

(Advanced Nodes Only) The constraint applies only to the edges in the specified direction. The default is 'any.'

Type: Boolean

'allCorner | 'concaveCorner | 'convexCorner | 'mixedCorner

(Advanced Nodes Only) 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

'exceptExactLength (*f\_exactLength f\_adjLength*)

(Advanced Nodes Only) The constraint does not apply if an edge has length exactly equal to *exactLength* and has an adjacent edge with length less than or equal to *adjLength*.

'exceptExactBothLength (*f\_exactLength f\_exactAdjLength*)

(Advanced Nodes Only) The constraint does not apply if an edge has length exactly equal to *exactLength* and has an adjacent edge with length exactly equal to *exactAdjLength*.

## Examples

- [Example 1: minStepEdgeLength with horizontalEdge, convexCorner, and mixedCorner](#)
- [Example 2: minStepEdgeLength with exceptExactLength and exceptExactBothLength](#)

## Virtuoso Technology Data Constraint Reference

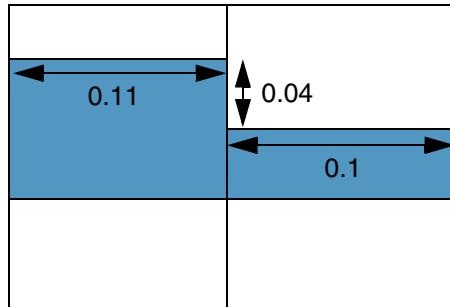
### Length and Width Constraints

#### ***Example 1: 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 Constraint Reference

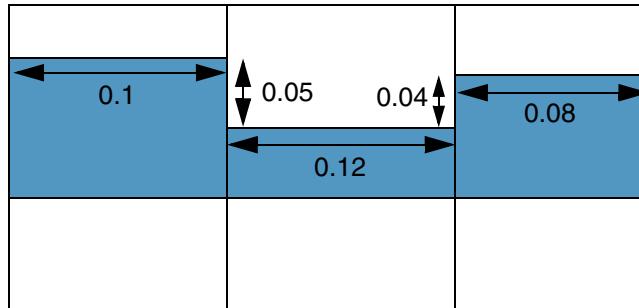
### Length and Width Constraints

#### ***Example 2: 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 exactly equal to 0.04.

```
spacings(
  ( minStepEdgeLength "Metal1"
    'horizontalEdge
    'exceptExactLength ((0.1 0.06))
    'exceptExactBothLength ((0.08 0.04))
    0.12
  )
) ; spacings
```

 Metal1



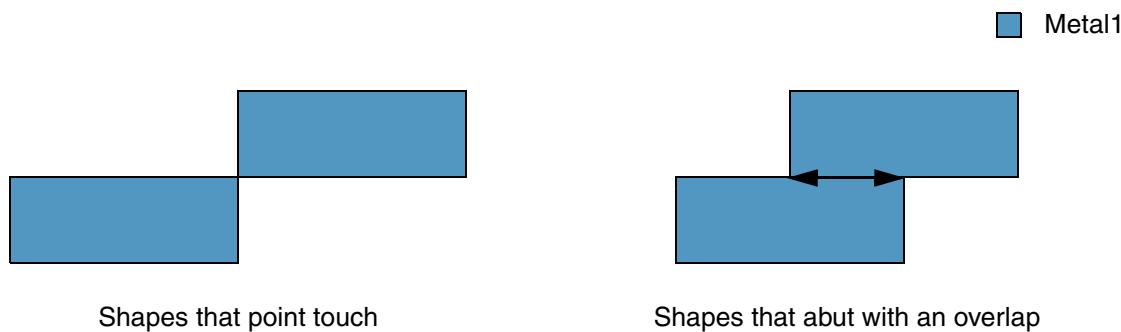
PASS. The left horizontal edge has an adjacent edge with length 0.05 (<0.06), and is, therefore, exempt; the middle horizontal edge is 0.12 long, and, therefore, satisfies the constraint; the right horizontal edge has an adjacent edge that is exactly equal to 0.04, and is, therefore, also exempt.

## minWidth

```
spacings(
  ( minWidth tx_layer
    ['exceptPointTouch 'exceptLineTouch (g_range) ]
    ['vertical | 'horizontal]
    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 Constraint 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

'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 Generate All From Source and Update Components And Nets commands.

## Virtuoso Technology Data Constraint Reference

### Length and Width Constraints

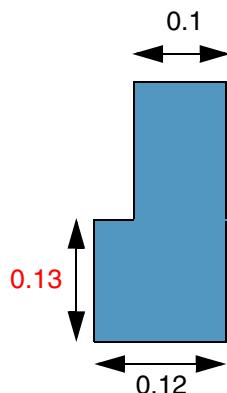
---

#### Example

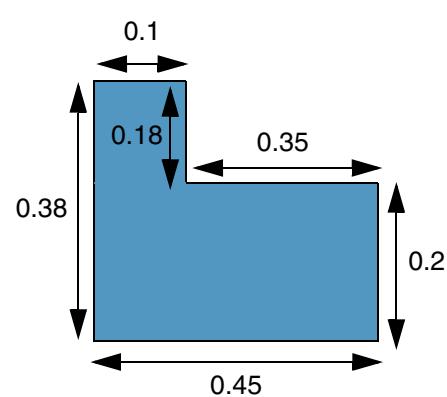
The width of a shape when measured horizontally must be at least 0.1 and when measured vertically must be at least 0.2.

```
spacings(
  ( minWidth "Metal1"
    'horizontal
    0.1
  )
  ( minWidth "Metal1"
    'vertical
    0.2
  )
) ;spacings
```

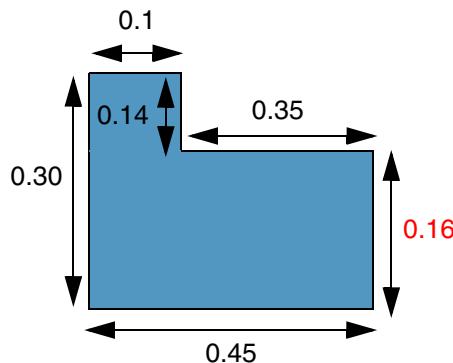
 Metal1



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.

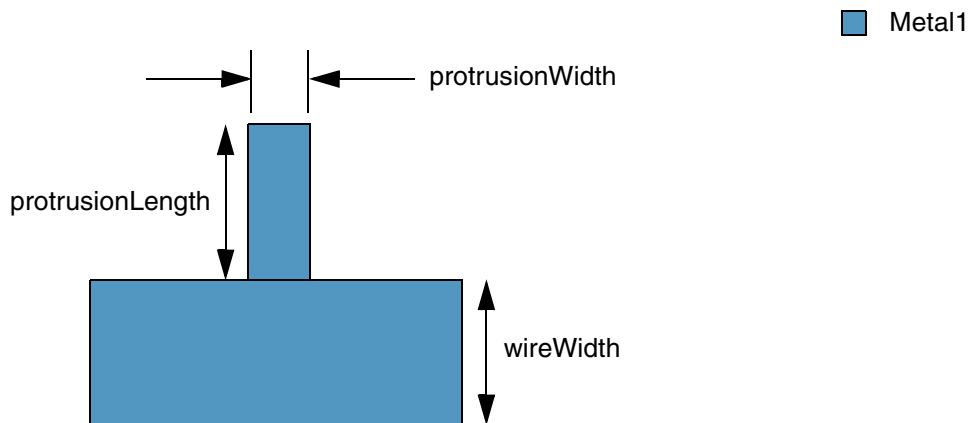


c) FAIL. The 0.16 width measured vertically is less than the required minimum width of 0.2 in that direction.

## 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.

## 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 chapter includes the following constraints:

- [allowedShapeAngles](#)
- [allowedSnapPatternDefs \(ICADV12.3 Only\)](#)
- [allowedWireTypes \(ICADV12.3 Only\)](#)
- [diagonalShapesAllowed](#)
- [edgeMustCoincide \(Advanced Nodes Only\)](#)
- [edgeMustOverlap \(Advanced Nodes Only\)](#)
- [errorLayer](#)
- [gateOrientation](#)
- [maxACCCurrentDensity](#)
- [maxNumCorners \(Advanced Nodes Only\)](#)
- [minOneDArrayStructure](#)
- [minStitchOverlap \(Advanced Nodes Only\)](#)
- [msCrossTieLayers](#)
- [msLayerCrossTieInterval](#)
- [msShieldingLimit](#)
- [msShieldStyle](#)
- [multiMaskCheck \(ICADV12.3 Only\)](#)
- [rectShapeDir](#)
- [sameMaskOnLayer \(ICADV12.3 Only\)](#)
- [trimMetalTrack \(ICADV12.3 Only\)](#)

## **Virtuoso Technology Data Constraint Reference**

### Miscellaneous Constraints

---

- [trimShape \(ICADV12.3 Only\)](#)
- [validCutClass](#)
- [vertexInsideForbidden \(ICADV12.3 Only\)](#)

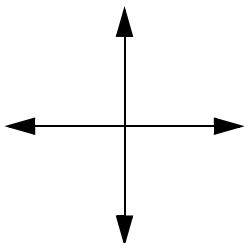
## allowedShapeAngles

```
allowedShapeAngles(  
    ["anyAngle" | "orthogonal" | "diagonal"]  
    (tx_layer {"anyAngle" | "orthogonal" | "diagonal"})  
    ...  
) ;allowedShapeAngles
```

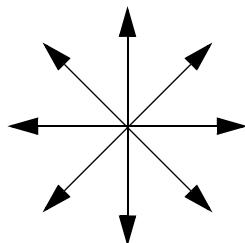
Sets the angle or angles at which a shape on a specified layer can be placed. It also allows specification of the default angles for layers not specifically assigned angles.

The arrows illustrate the angles allowed by each setting.

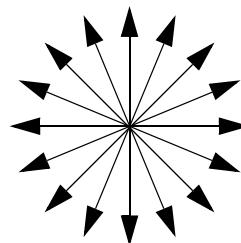
orthogonal allows vertical and horizontal shapes.



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)

## 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 (ICADV12.3 Only)**

```
spacings(
  ( allowedSnapPatternDefs
    (t_snapPatternDef ...)
  )
) ;spacings
```

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 (ICADV12.3 Only)**

```
spacings(
  ( allowedWireTypes tx_layer
    (t_wireType ...)
  )
) ;spacings
```

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.

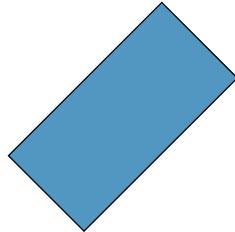
```
spacings(
  ( allowedWireTypes "Metal3"
    ("1X" "2X" "3X")
  )
) ;spacings
```

## 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

<i>tx_layer</i>	The layer on which the constraint is applied.  Type: String (layer and purpose names) or Integer (layer number)
t   nil	If t, shapes can be placed diagonally at 45 degrees. If nil, 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 (Advanced Nodes Only)**

```
orderedSpacings(  
    ( edgeMustCoincide tx_layer1 tx_layer2  
        [ 'horizontalEdge' | 'verticalEdge'  
        [ 'edgeLengthRanges (g_ranges) [ 'endOfLineOnly ]  
    )  
) ;orderedSpacings
```

Specifies the *layer1* edges that must coincide with a *layer2* edge.

## Values

*tx\_layer1* The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

## Parameters

'horizontalEdge | 'verticalEdge

The constraint applies only to *layer1* edges that are in this direction.

Type: Boolean

'edgeLengthRanges (g\_ranges)

The constraint applies only if the `layer1` edge length is in this range.

Type: Floating-point values specifying the length ranges to which the constraint applies.

'endOfLineOnly

The constraint applies only if the `layer1` edge is between two convex corners and its length falls in the range specified with '`edgeLengthRanges`'.

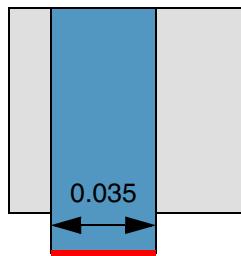
Type: Boolean

## Example

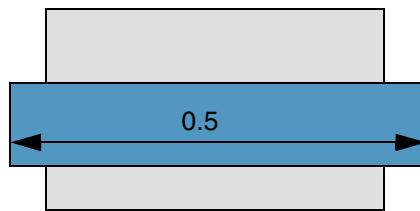
Horizontal Metal1 edges between two convex corners must coincide with a Metal2 edge if the edge length is less than 0.02 or greater than 0.03 and less than 0.04.

```
orderedSpacings(
  ( edgeMustCoincide "Metal1" "Metal2"
    'horizontalEdge
    'edgeLengthRanges ( "<0.02" " (0.03 0.04) "
    'endOfLineOnly
  )
) ;orderedSpacings
```

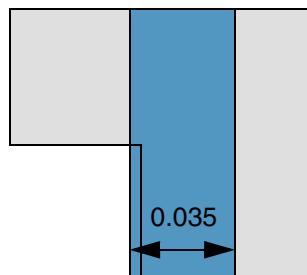
 Metal2  
 Metal1



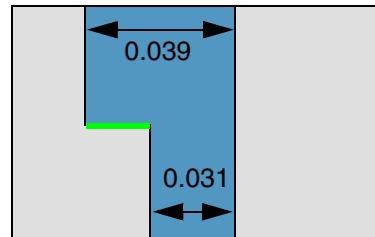
a) FAIL. The Metal1 horizontal edges are 0.035 long, a value that falls in the specified range. Therefore, the constraint applies, but is not met because the bottom edge is not coincident with a Metal2 edge.



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 (Advanced Nodes Only)**

```
orderedSpacings(  
    ( edgeMustOverlap tx_layer1 tx_layer2  
        [ 'horizontalEdge' | 'verticalEdge'  
        [ 'edgeLengthRanges (g_ranges) [ 'endOfLineOnly]  
    )  
) ;orderedSpacings
```

Species 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

*tx\_layer1* The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

## Parameters

'horizontalEdge | 'verticalEdge

The constraint applies only to *layer1* edges that are in this direction.

Type: Boolean

'edgeLengthRanges (g\_ranges)

The constraint applies only if the `layer1` edge length is in this range.

Type: Floating-point values specifying the length [ranges](#) to which the constraint applies.

## Virtuoso Technology Data Constraint Reference

### Miscellaneous Constraints

---

'endOfLineOnly

The constraint applies only if the *layer1* edge is between two convex corners and its length falls in the range specified with 'edgeLengthRanges.

Type: Boolean

### Examples

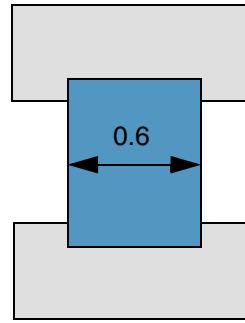
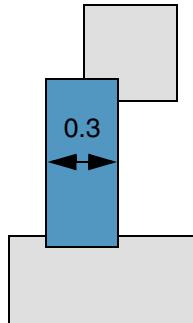
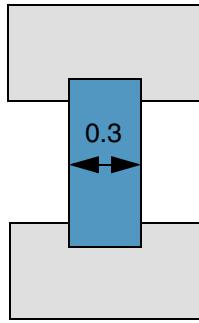
- [Example 1: edgeMustOverlap with horizontalEdge and edgeLengthRanges](#)
- [Example 2: edgeMustOverlap with horizontalEdge, endOfLineOnly, and edgeLengthRanges](#)

**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
```

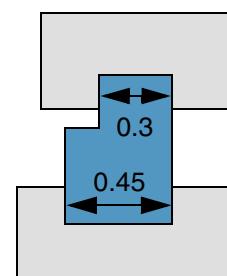
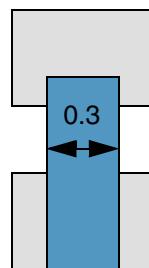
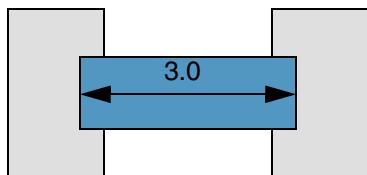
 Metal2  
 Metal1



a) PASS. The constraint applies to the Metal1 horizontal edges ( $\text{length} \leq 0.5$ ) 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).

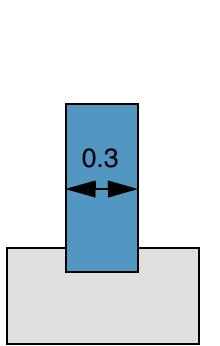
f) FAIL. The constraint applies, but is not met because the middle horizontal Metal1 edge is not fully overlapped by a Metal2 shape.

**Example 2: *edgeMustOverlap* with *horizontalEdge*, *endOfLineOnly*, and *edgeLengthRanges***

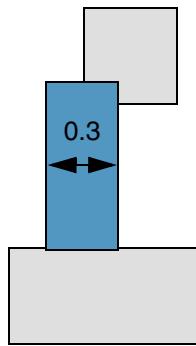
All horizontal Metal1 edges that are between two convex corners (end-of-line edges) and are less than or equal to 0.5 long must be fully overlapped by a Metal2 shape.

```
orderedSpacings(
    (edgeMustOverlap "Metal1" "Metal2"
        'horizontalEdge
        'edgeLengthRanges "<=0.5"
        'endOfLineOnly
    )
) ;orderedSpacings
```

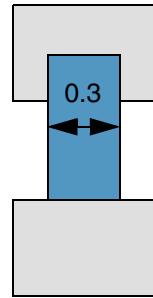
 Metal2  
 Metal1



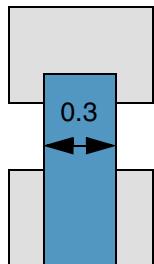
a) FAIL. The top Metal1 edge is not overlapped by a Metal2 shape.



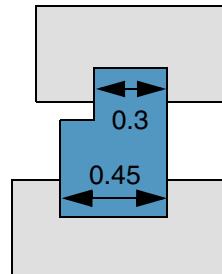
b) FAIL. The top Metal1 edge is not fully overlapped by the Metal2 shape.



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
```

## 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

*tx\_derivedLayer*      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

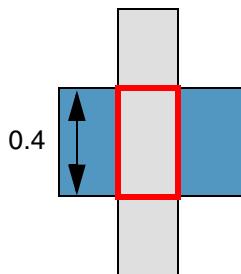
*f\_width*      The constraint applies only to the gates that have width less than or equal to this value. Otherwise, the constraint applies to gates of any width.

## Example

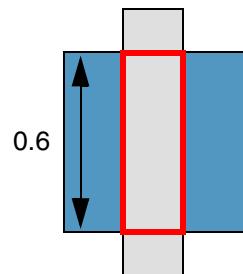
Orientation must be horizontal for gates with width less than or equal to 0.5.

```
spacings(
  ( gateOrientation "Gate1"
    'width 0.5
    "horizontal"
  )
) ;spacings
```

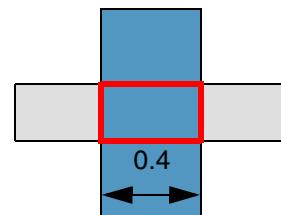
<span style="background-color: #e0e0e0; border: 1px solid black; display: inline-block; width: 1em; height: 1em;"></span>	Poly
<span style="background-color: #0070C0; border: 1px solid black; display: inline-block; width: 1em; height: 1em;"></span>	Diffusion
<span style="background-color: red; border: 1px solid black; display: inline-block; width: 1em; height: 1em;"></span>	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.

## **maxACCCurrentDensity**

```
layerRules(
    currentDensity(
        ( rmsACCCurrentDensity tx_layer f_value )
    ) ;currentDensity

    currentDensityTables(
        ( rmsACCCurrentDensity tx_metalLayer
            (( "frequency" nil nil ["width" nil nil] )
            ( g_table )
        )
        ( rmsACCCurrentDensity tx_cutLayer
            (( "frequency" nil nil ["cutArea" nil nil] )
            ( g_table )
        )
    )
) ;currentDensityTables

) ;layerRules

spacings(
    ( maxACCCurrentDensity tx_layer
        {'peak' | 'rms' | 'average'}
        ['tempFactor ( (( "temperature" )) (f_temp f_mult ...) )]
        ['hoursFactor ( (( "hours" )) (f_hour f_mult ...) )]
        f_value
    )
)

) ;spacings

spacingTables(
    ( maxACCCurrentDensity 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)
) ;maxACCCurrentDensity

( maxACCCurrentDensity 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)
) ;maxACCCurrentDensity

) ;spacingTables
```

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*.

*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 actual 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 actual 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 actual 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 Constraint Reference

### Miscellaneous Constraints

---

'hoursFactor ( (( "hours" )) ( *f\_hour f\_mult ...* ) )

A multiplier table that is indexed on the number of hours of operation. When the actual 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 ( $7.5 * 1.8 * 0.75$ ).

```
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 (Advanced Nodes Only)**

```
spacings(
  ( maxNumCorners tx_layer
    ['allCorner | 'concaveCorner | 'convexCorner]
    x_count
  ) ;spacings
```

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 perfectly 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 perfectly 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, perfectly 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 Constraint Reference

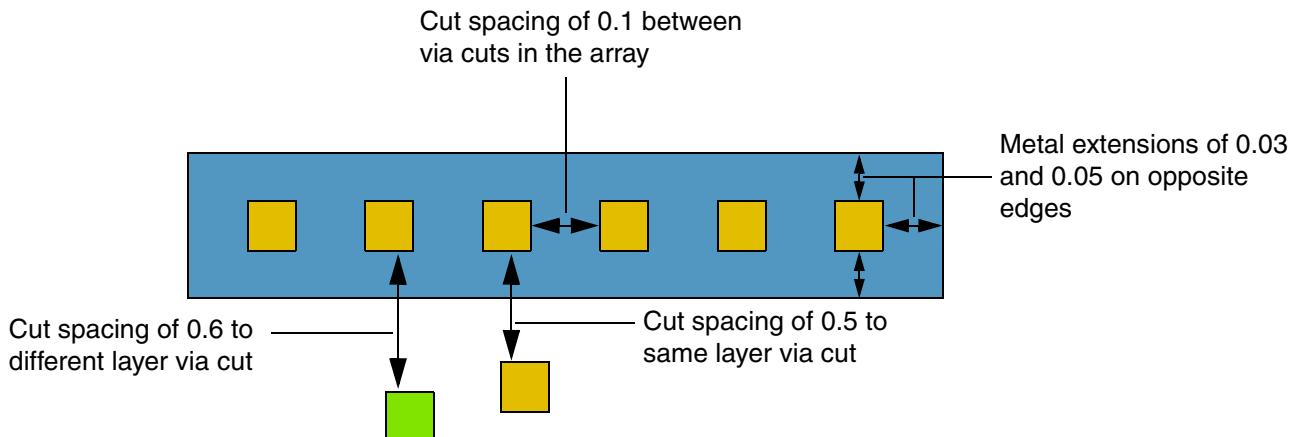
### Miscellaneous Constraints

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" "Metall1" "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 (Advanced Nodes Only)

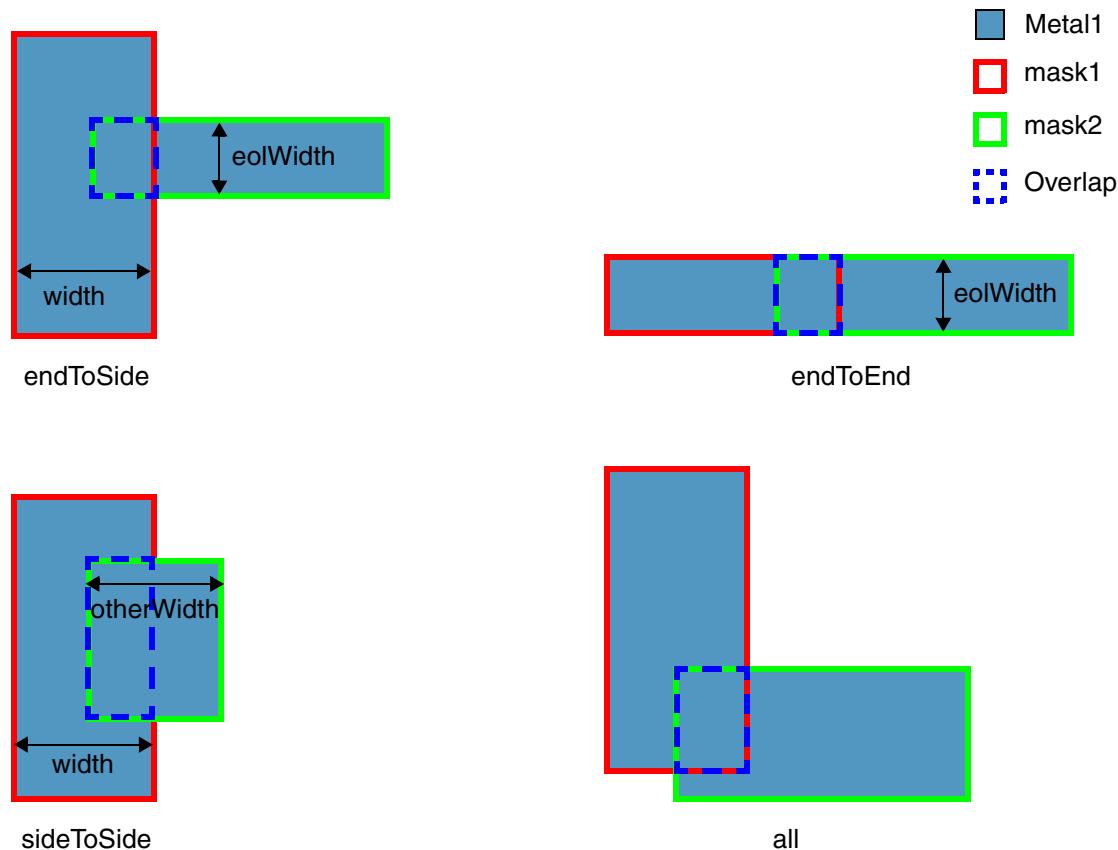
```
spacings(
    ( minStitchOverlap tx_layer
        [ 'all | 'endToEnd | 'endToSide | 'endToAny | 'sideToSide]
        [ 'eolWidth f_eolWidth]
        [ 'width f_width [ 'otherWidth f_otherWidth] ]
        (f_overlapValue)
    )
)

) ; spacings
```

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.



## 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

'all | 'endToEnd | 'endToSide | 'endToAny | 'sideToSide

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

'eolWidth *f\_eolWidth*

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.

'width *f\_width*

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

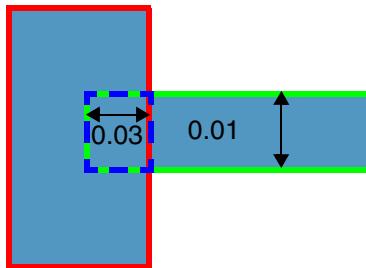
- [Example 1: minStitchOverlap with endToSide and eolWidth](#)
- [Example 2: minStitchOverlap with endToEnd and eolWidth](#)

### ***Example 1: minStitchOverlap with endToSide and eolWidth***

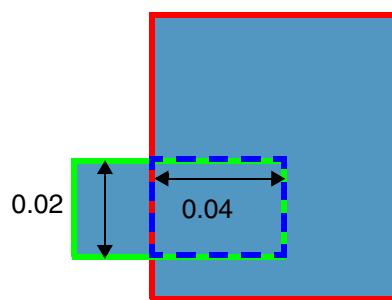
The end-to-side overlap between a Metal1 end-of-line edge less than 0.022 wide and a Metal1 side on different masks must be at least 0.04.

```
spacings(
  ( minStitchOverlap "Metal1"
    'endToSide
    'eolWidth 0.022
    (0.04)
  )
) ;spacings
```

<span style="background-color: #005293; border: 1px solid black; width: 10px; height: 10px;"></span>	Metal1
<span style="background-color: red; border: 1px solid black; width: 10px; height: 10px;"></span>	mask1
<span style="background-color: green; border: 1px solid black; width: 10px; height: 10px;"></span>	mask2
<span style="border: 2px dashed blue; width: 10px; height: 10px;"></span>	Overlap



a) FAIL. The end-of-line width is 0.01 (<0.022), but the overlap between the shapes is 0.03 (<0.04).



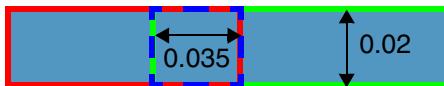
b) PASS. The end-of-line width is 0.02 (<0.022) and the overlap between the shapes is 0.04.

**Example 2: *minStitchOverlap* with *endToEnd* and *eolWidth***

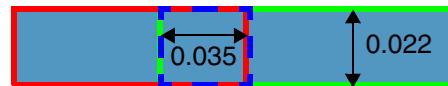
The end-to-end overlap between two Metal1 end-of-line edges less than 0.022 wide on different masks must be at least 0.04.

```
spacings(
  ( minStitchOverlap "Metal1"
    'endToEnd
    'eolWidth 0.022
    (0.04)
  )
) ;spacings
```

<span style="color: blue;">█</span> Metal1 <span style="color: red;">█</span> mask1 <span style="color: green;">█</span> mask2 <span style="border: 1px dashed blue; padding: 2px;">█</span> 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
```

## **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
```

## **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.

<b>Value</b>	<b>Name</b>
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 Constraint 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 (ICADV12.3 Only)**

```
spacings(
  ( multiMaskCheck tx_layer
    x_value
  )
) ;spacings
```

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 ' <code>'sameMask</code> ' parameter. If the value is 3, shapes on the layer are checked for wheel configuration violations based on the spacing rules specified with the ' <code>'sameMask</code> ' parameter.  <b>Note:</b> 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
```

## rectShapeDir

```
spacings(
    ( rectShapeDir tx_layer
        ['extendBy f_extendBy ['widthRanges (g_ranges)']]
        ['exceptViaLayer tx_viaLayer
            'exceptViaSize (f_viaSize1 f_viaSize2)
            'exceptExactSize (f_exactMetalSize1 f_exactMetalSize2)
        ]
        ['exceptEdgeLength f_maxLength 'width f_maxWidth ['twoSides]]
        ['exceptRanges (g_exceptRanges)]
        {"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)
        ]
        ['exceptEdgeLength f_maxLength 'width f_maxWidth ['twoSides]]
        ['exceptRanges (g_exceptRanges)]
        [ "any" | "horizontal" | "vertical" ]
    )
    (g_table)
)
)

) ;spacingTables
```

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.

## Values

<code>tx_layer</code>	The layer on which the constraint is applied.  Type: String (layer and purpose names) or Integer (layer number)
<code>"any"   "horizontal"   "vertical"</code>	The direction of the rectangle.  Type: Boolean
<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:  $f_width \{ "any"   "horizontal"   "vertical" \}$ where,  $f_width$ 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>	(Advanced Nodes Only) All end-of-line edges must be extended by this value before the constraint is applied.
<code>'widthRanges g_ranges</code>	(Advanced Nodes Only) 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.

```
'exceptViaLayer tx_viaLayer
'exceptViaSize f_viaSize1 f_viaSize2
'exceptExactSize f_exactMetalSize1 f_exactMetalSize2
```

(Advanced Nodes Only) 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.

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

```
'exceptEdgeLength f_maxLength
```

(Advanced Nodes Only) The constraint does not apply if the rectangular shape has a jog with width less than or equal to this value.

```
'width f_maxWidth
```

(Advanced Nodes Only) 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
```

(Advanced Nodes Only) 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)
```

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.

## Examples

- [Example 1: rectShapeDir with vertical](#)
- [Example 1: rectShapeDir with vertical](#)
- [Example 3: rectShapeDir with extendBy and widthRanges](#)

- [Example 4: rectShapeDir with exceptEdgeLength, width, and twoSides](#)
- [Example 5: rectShapeDir with exceptRanges](#)

***Example 1: rectShapeDir with vertical***

A shape on layer Metal1 must be a rectangle in the vertical direction.

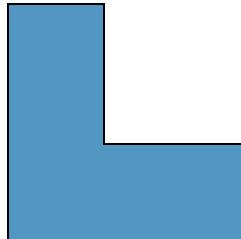
```
spacings(
  ( rectShapeDir "Metal1"
    "vertical"
  )
) ; spacings
```

 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.

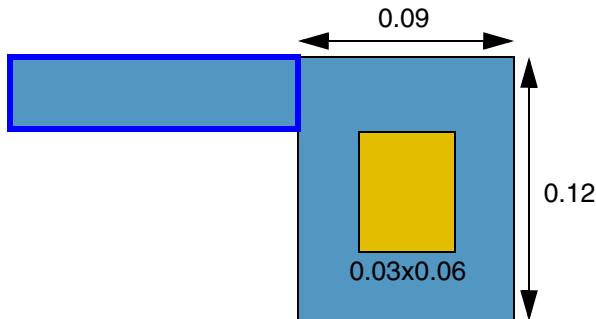
d) FAIL. The shape is a horizontal rectangle.

**Example 2: rectShapeDir with exceptViaLayer, exceptViaSize, and exceptExactSize**

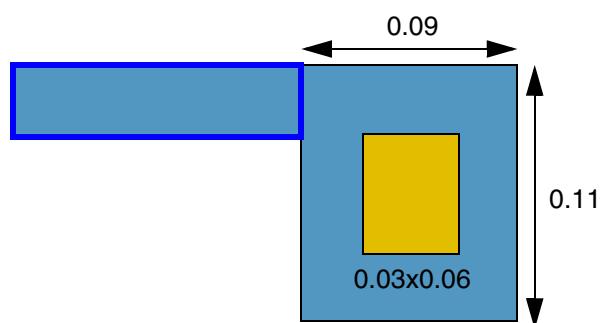
A Metal1 shape with dimensions 0.09x0.12 is overlapped by a 0.3x0.06 via. The constraint is met if the remaining portion of the metal shape is a horizontal rectangle, exactly aligned with a horizontal edge of the exempted portion of the metal shape.

```
spacings(
  ( rectShapeDir "Metal1"
    'exceptViaLayer "Via1"
    'exceptViaSize (0.03 0.06)
    'exceptExactSize (0.09 0.12)
    "horizontal"
  )
) ; spacings
```

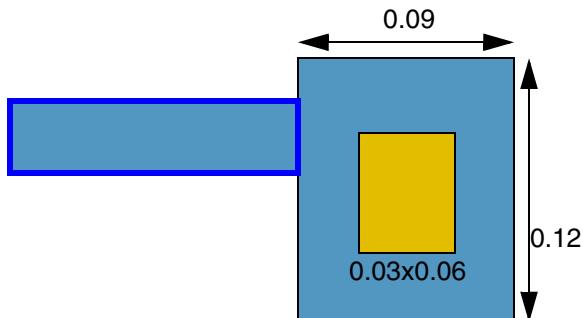
 Via1  
 Metal1



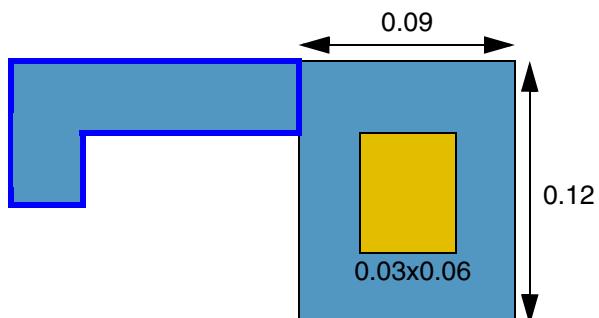
a) PASS. The 0.09x0.12 metal shape is overlapped by a 0.03x0.06 via on layer Via1, and is, therefore, exempt. The remaining metal shape (with a thick blue border) is a horizontal rectangle.



b) FAIL. The 0.09x0.11 metal shape does not meet the 'exceptExactSize' requirement ( $0.11 < 0.12$ ).



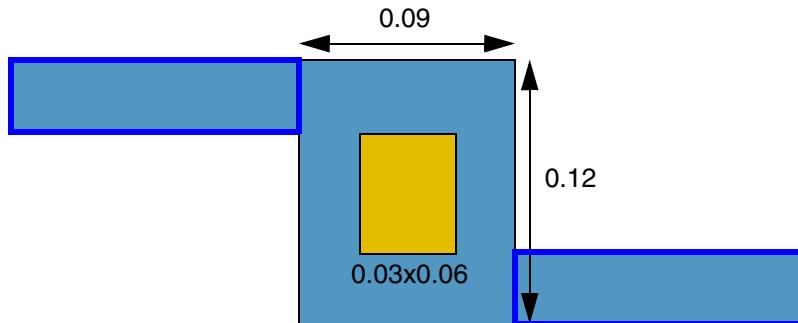
c) FAIL. The 0.09x0.12 metal shape is overlapped by a 0.03x0.06 via on layer Via1, and is, therefore, exempt. The remaining metal shape (with a thick blue border) is a horizontal rectangle, but is not exactly aligned with a horizontal edge of the exempted portion of the metal shape.



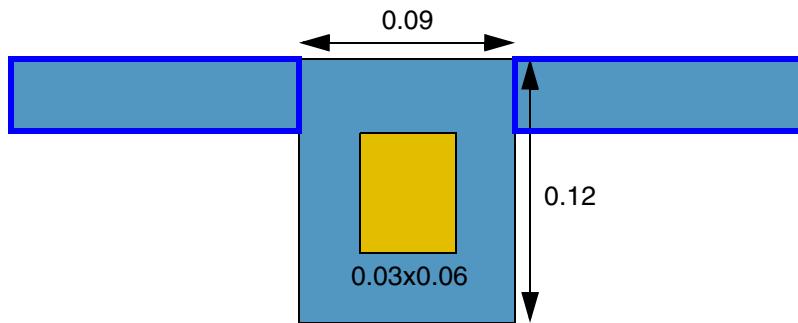
d) FAIL. The 0.09x0.12 metal shape is overlapped by a 0.03x0.06 via on layer Via1, and is, therefore, exempt. However, the remaining metal shape (with a thick blue border) is not a rectangle.

## Virtuoso Technology Data Constraint Reference

### Miscellaneous Constraints



e) FAIL: The  $0.09 \times 0.12$  metal shape is overlapped by a  $0.03 \times 0.06$  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.



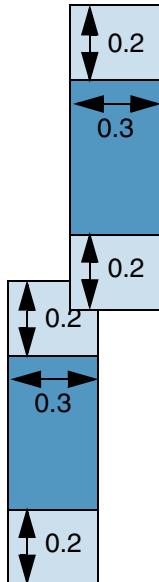
f) PASS: The  $0.09 \times 0.12$  metal shape is overlapped by a  $0.03 \times 0.06$  via on layer Via1, and is, therefore, exempt. The remaining metal shapes (with a thick blue border) are both rectangles and are exactly aligned with the same horizontal edge of exempted portion of the metal shape.

**Example 3: rectShapeDir with extendBy and widthRanges**

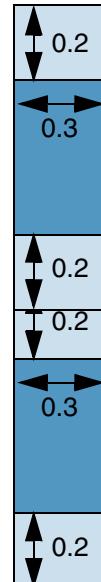
When the edges of a rectangular Metal1 shape with width less than or equal to 0.3 or equal to 0.6 or 0.8 wide are extended by 0.2, the resulting shape must also be rectangular.

```
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 Constraint Reference

### Miscellaneous Constraints

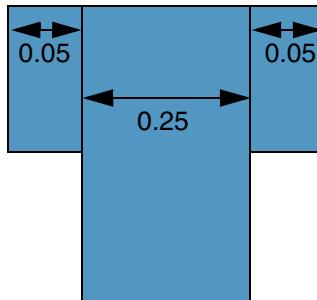
---

#### **Example 4: *rectShapeDir* with *exceptEdgeLength*, *width*, and *twoSides***

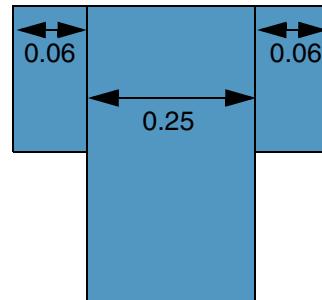
A vertical rectangle on layer Metal1 with width 0.25 must have 0.05 wide jogs on both sides.

```
spacings(
  ( rectShapeDir "Metal1"
    'exceptEdgeLength 0.05 'width 0.25 'twoSides
    "vertical"
  )
) ; spacings
```

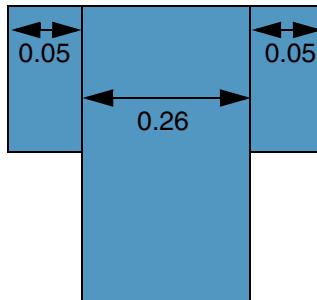
Metal1



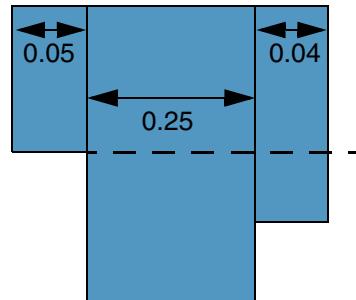
a) PASS. The two jogs are each 0.05 wide and the width of the vertical rectangle is 0.25.



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
```

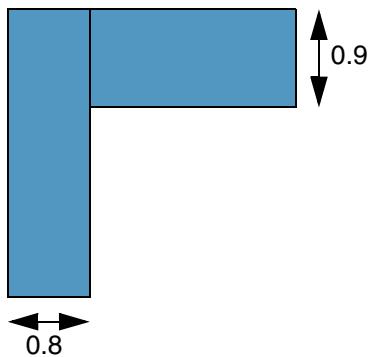
 Metal1



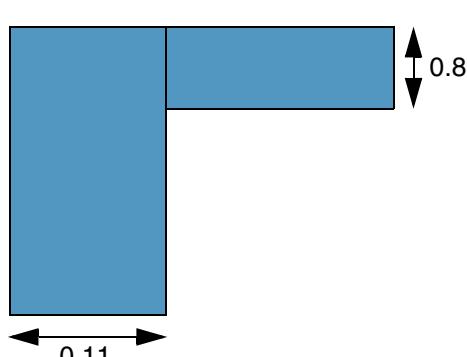
a) FAIL. The constraint applies because the width of the rectangle is not in the exempted width range. However, the rectangle is horizontal.



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 (ICADV12.3 Only)**

```
spacings(
  ( sameMaskOnLayer tx_layer
    [ 'mask1 | 'mask2 | 'mask3 | 'mask4]
  )
) ;spacings
```

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**

'mask1   'mask2   'mask3   'mask4	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 (ICADV12.3 Only)**

```
spacings(
  ( trimMetalTrack tx_trimLayer
    'coreOffset f_offset
    ['mask1 | 'mask2 | 'mask3]
    f_pitch
  )
) ;spacings
```

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**

*tx\_trimLayer*      The layer on which the constraint is applied.

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

*f\_pitch*      The pitch value.

### **Parameters**

'coreOffset *f\_offset*

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

## Virtuoso Technology Data Constraint Reference

### Miscellaneous Constraints

---

#### Example

The tracks on TM1 are defined on mask1, with 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
```

## trimShape (ICADV12.3 Only)

```
spacings(
  ( trimShape tx_layer
    'exactWidth f_exactWidth
    ['maxLength f_maxLength]
    ['exceptSpacing f_spacing]
    ['exceptWidth f_width]
    ['extension f_extension ['metalEdge]]
    ['mask1 | 'mask2 | 'mask3 | 'mask4]
    {"adjacentTrack" | "midTrack" | "extension"}
  )
) ;spacings
```

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

*tx\_layer*                          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 perfectly aligned.

## Parameters

'exactWidth *f\_exactWidth*

The width of the shape formed on the trim metal layer must be exactly 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.

'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

## Examples

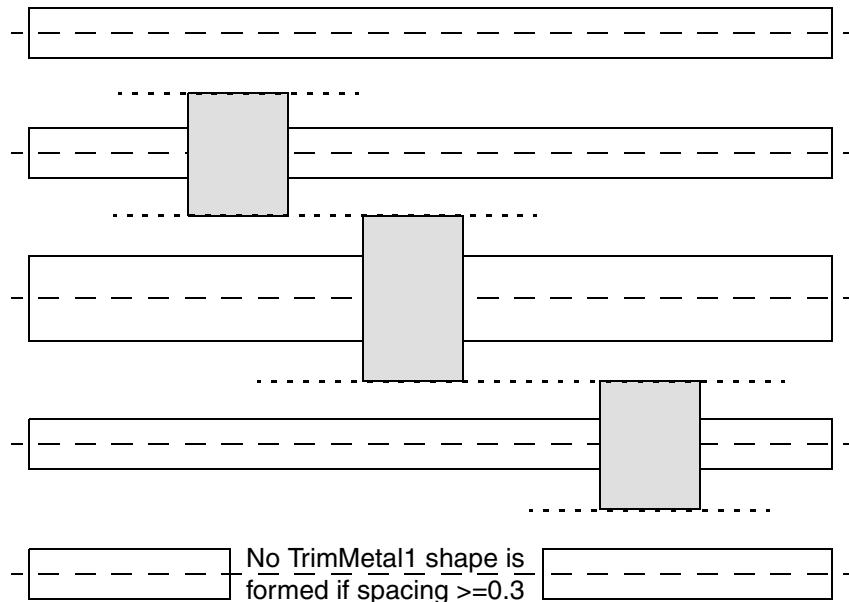
- [Example 1: trimShape with midTrack](#)
- [Example 2: trimShape with adjacentTrack](#)
- [Example 3: trimShape with extension](#)
- [Example 4: trimShape with extension and metalEdge](#)

**Example 1: trimShape with midTrack**

The width of the TrimMetal1 shapes must be exactly 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



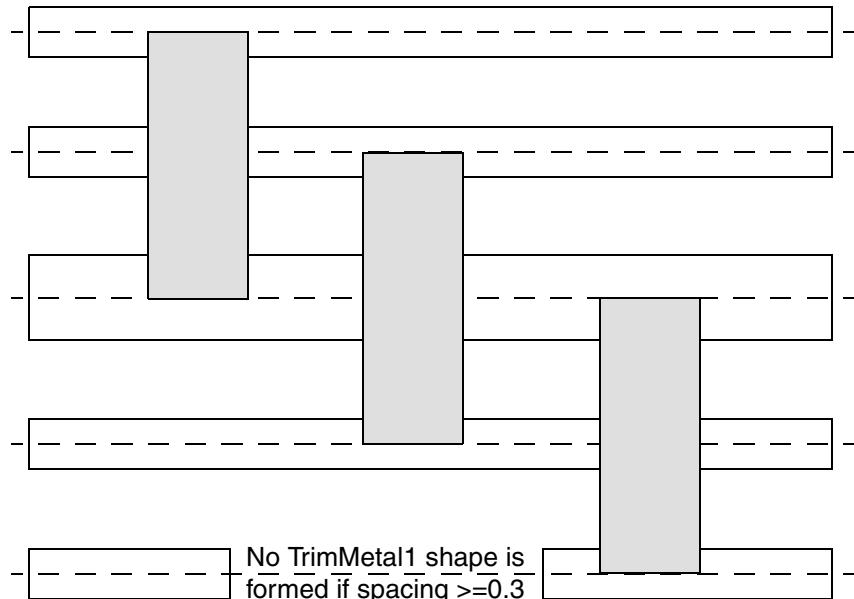
No TrimMetal1 shape is formed if spacing >=0.3

**Example 2: trimShape with adjacentTrack**

The width of each TrimMetal1 shape must be exactly 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

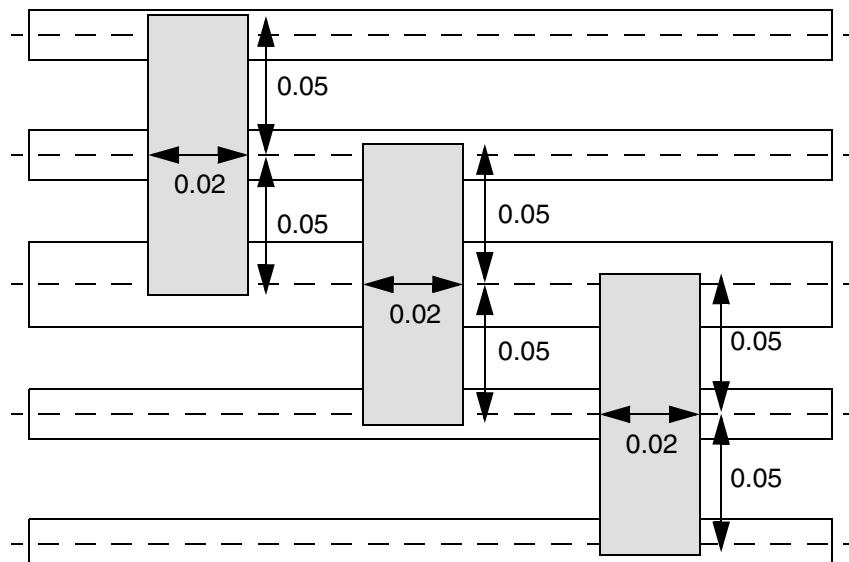


**Example 3: trimShape with extension**

The width of each TrimMetal1 shape must be exactly 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

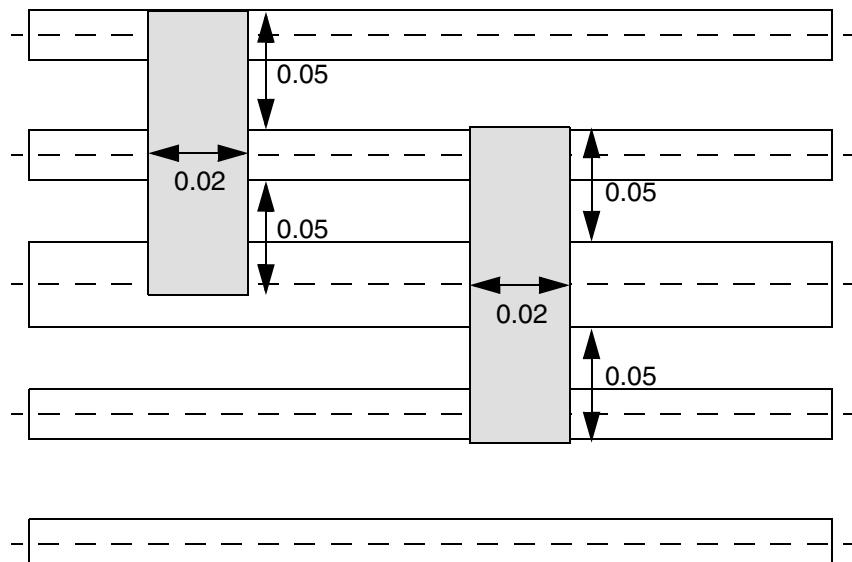


**Example 4: trimShape with extension and metalEdge**

The width of each TrimMetal1 shape must be exactly 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



## **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**

*tx\_cutLayer*      The cut layer on which the constraint is applied.

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

*(f\_width f\_length) | t\_name*

The cut class allowed for the specified layer, specified 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

### **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.

## 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 (ICADV12.3 Only)**

```
orderedSpacings(  
    ( vertexInsideForbidden tx_layer1 tx_layer2 )  
) ;orderedSpacings
```

Specifies that the vertices of a shape on *layer1* are not allowed inside a shape on *layer2*.

### **Values**

*tx\_layer1*      The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

### **Parameters**

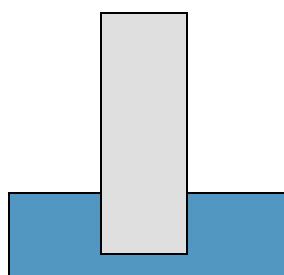
None

### **Example**

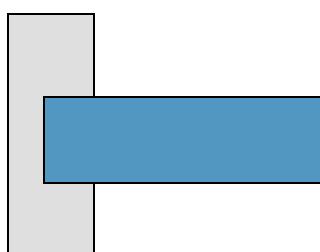
The constraint allows Metal2 vertices inside a Metal1 shape. However, Metal1 vertices are not allowed inside a Metal2 shape.

```
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.

**Virtuoso Technology Data Constraint Reference**  
Miscellaneous Constraints

---

---

## NumCut Constraints

---

This chapter includes the following constraints:

- [maxViaClusterNumCuts](#)
- [minNumCut](#)
- [minProtrusionNumCut](#)

## 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.

## 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.

```
'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

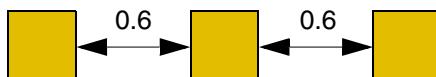
- [Example 1: maxViaClusterNumCuts with cutClass, within, and diagonalNumCut](#)
- [Example 2: maxViaClusterNumCuts with cutClass, within, diagonalNumCut, diagonalSpacing, diagonalWithin, sideExtension, and edgeExtension](#)

**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:

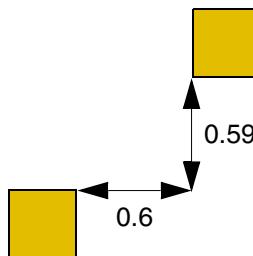
- The distance between via cuts must be greater than or equal to 0.6 and less than or equal to 0.7.
- At most three via cuts can be placed along a straight diagonal line.

```
spacings(
  ( maxViaClusterNumCuts "Via1"
    'cutClass 0.3
    'within (0.6 0.7)
    'diagonalNumCut 3
    3
  )
) ;spacings
```

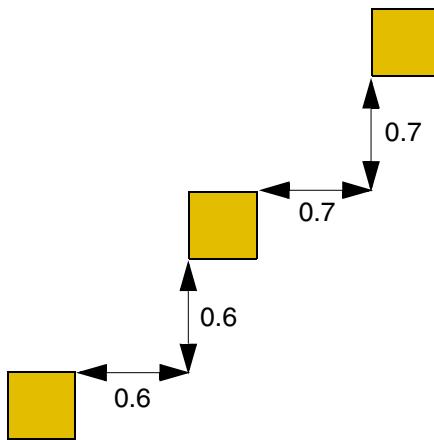


a) PASS. The three via cuts are perfectly aligned and are at a distance 0.6 from each other.

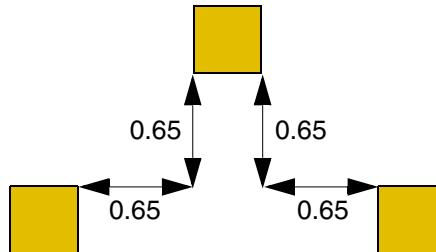
■ Via1



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).

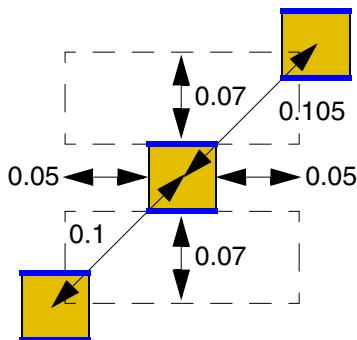
**Example 2: maxViaClusterNumCuts with cutClass, within, diagonalNumCut, diagonalSpacing, diagonalWithin, sideExtension, and edgeExtension**

The number of 0.3x0.3 Via1 via cuts in a cluster cannot exceed 3. The following additional conditions must also be met:

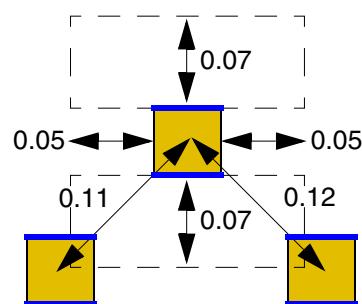
- The via cuts when placed along a straight diagonal line must be within a distance of 0.13 from each other and the spacing between them must be greater than or equal to 0.1.
- The via cuts when placed in a bend or notch formation must be within a distance of 0.13 from each other and the spacing between 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
    )
)
;spacings
```

█ Via1  
 Search window



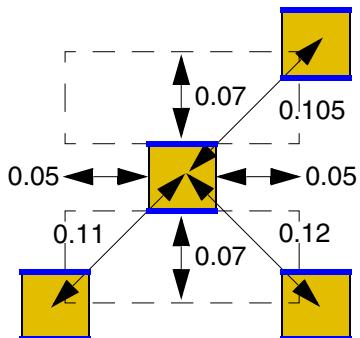
a) PASS. The via cuts are placed along a straight diagonal line within a distance of 0.13 from each other. Additionally, the via cuts overlap the search window and the spacing between the via cuts is greater than or equal to 0.1.



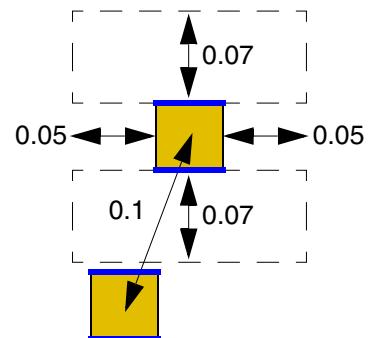
b) PASS. The via cuts are placed along a bend or notch within a distance of 0.13 from each other. Additionally, the via cuts overlap the search window and the spacing between the via cuts is greater than or equal to 0.11.

## Virtuoso Technology Data Constraint 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.

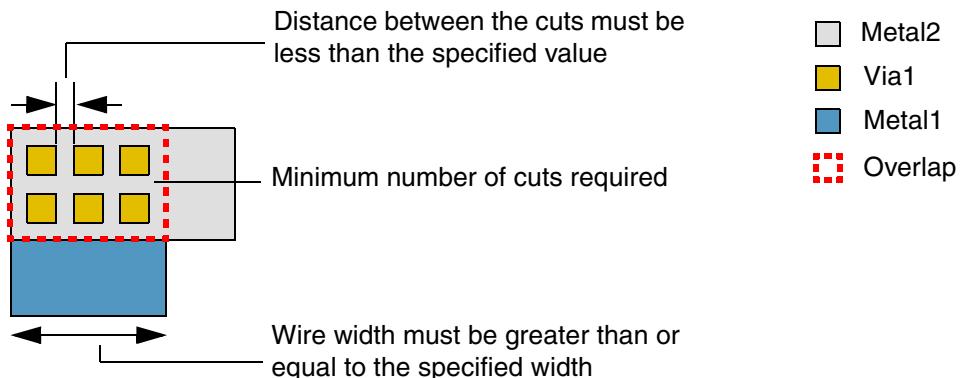
## minNumCut

```

spacingTables(
    ( minNumCut tx_cutLayer
        (( "width" nil nil)
         ['cutClass {f_cutWidth | (f_cutWidth f_cutLength) | t_name}]
         ['distanceWithin f_within] ['above | 'below | 'bothAboveBelow]
         ['sameMetalOverlap | 'fullyEnclosed]
         [f_default]
        )
        (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.
- **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

## Virtuoso Technology Data Constraint Reference

### NumCut Constraints

---

constraint group, including `minWidth`, `minNumCut`, `minSpacing`, `validLayers`, and `validVias`.

#### 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`

This identifies the index for `table`.

`g_table`      The format of the table is as follows:

`(f_width x_numberOfCuts)`

where,

- `f_width` 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.
- `x_numberOfCuts` 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

`'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_within`

The distance between the via cuts must be less than this value for the via cuts to be counted.

## Virtuoso Technology Data Constraint 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: (Advanced Nodes Only) 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: (Advanced Nodes Only) 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: (ICADV12.3 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.

## Examples

- [Example 1: minNumCut with above](#)
- [Example 2: minNumCut with bothAboveBelow](#)
- [Example 3: minNumCut with sameMetalOverlap](#)
- [Example 4: minNumCut with fullyEnclosed](#)

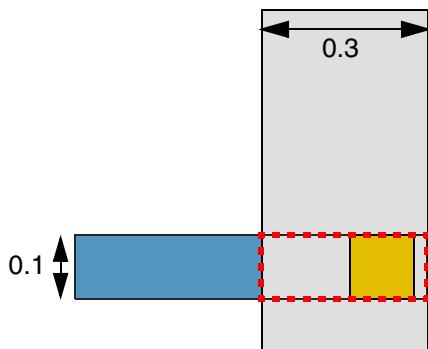
**Example 1: minNumCut with above**

The width of the wire on the layer above determines the number of cuts that a via must contain.

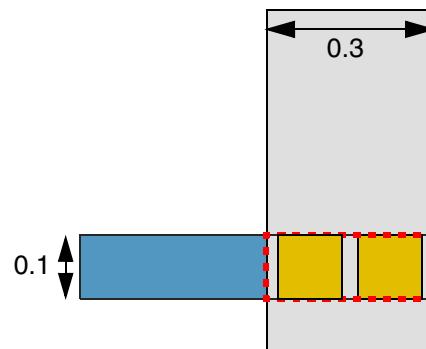
- Width  $\geq 0.1$ ; via cuts = 1
- Width  $\geq 0.3$ ; via cuts = 2
- Width  $\geq 0.5$ ; via cuts = 4
- Width  $\geq 0.6$ ; via cuts = 6

```
spacingTables(
  ( minNumCut "Via1"
    (( "width" nil nil)
     'above
    )
   (
    0.1 1
    0.3 2
    0.5 4
    0.6 6
   )
  )
) ;spacingTables
```

■	Metal2
■	Via1
■	Metal1
■	Overlap



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.

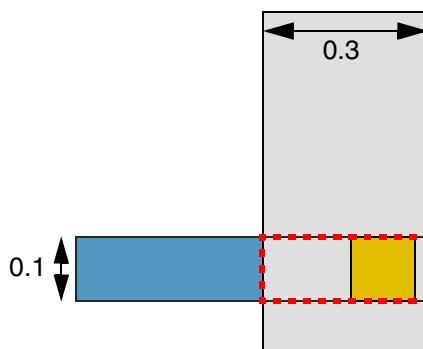
**Example 2: minNumCut with bothAboveBelow**

The width of the wires on the layers above and below determine the number of cuts that a via must contain.

- Width  $\geq 0.1$ ; via cuts = 1
- Width  $\geq 0.3$ ; via cuts = 2
- Width  $\geq 0.5$ ; via cuts = 4
- Width  $\geq 0.6$ ; via cuts = 6

```
spacingTables(
  ( minNumCut "Via1"
    ( ("width" nil nil)
      'bothAboveBelow
    )
    (
      0.1 1
      0.3 2
      0.5 4
      0.6 6
    )
  )
) ;spacingTables
```

- |   |         |
|---|---------|
| <span style="display: inline-block; width: 1em; height: 1em; background-color: #e0e0e0;"></span>  | Metal2  |
| <span style="display: inline-block; width: 1em; height: 1em; background-color: #cc9933;"></span>  | Via1    |
| <span style="display: inline-block; width: 1em; height: 1em; background-color: #336699;"></span>  | Metal1  |
| <span style="display: inline-block; width: 1em; height: 1em; border: 2px dashed #cc0000;"></span> | Overlap |



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***

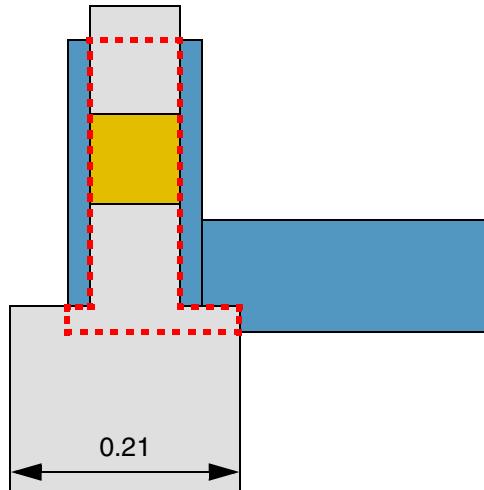
The width of the wider wire determines the number of cuts that a via must contain.

- 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
```

□	Metal2
■	Via1
■	Metal1
□	Overlap



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**

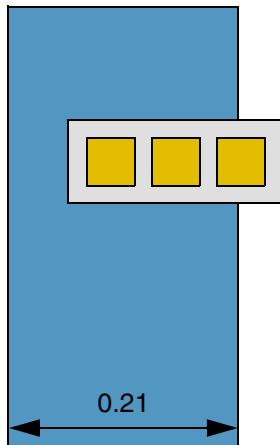
The width of the wider wire determines the number of cuts that a via must contain.

- 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

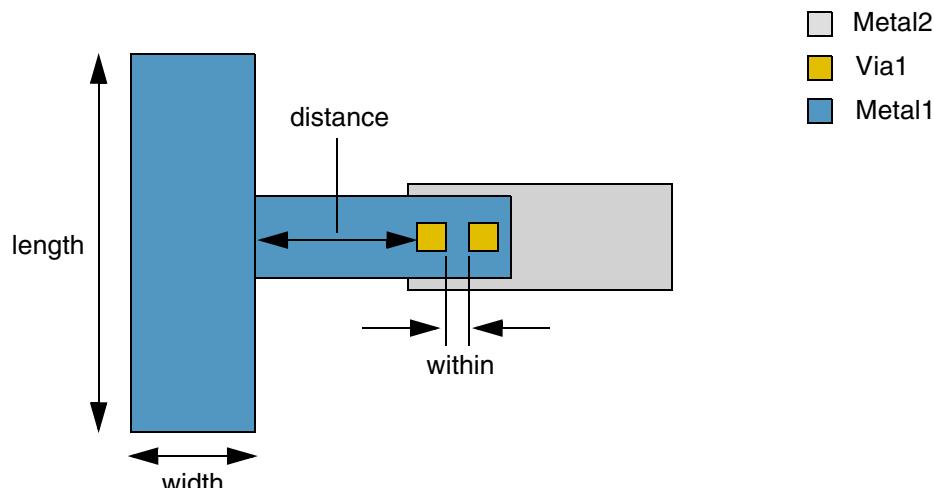
```

spacings(
    ( minProtrusionNumCut tx_cutLayer
        ['cutClass {f_width | (f_width f_length) | t_name}']
        'distance f_distance
        'width f_width
        'length f_length
        ['distanceWithin f_within]
        ['above | 'below]
        x_numCuts
    )
)
;spacings

spacings(
    ( minProtrusionNumCut tx_cutLayer
        ['cutClass {f_width | (f_width f_length) | t_name}']
        ['distance f_distance]
        'width f_width
        'area f_area
        ['distanceWithin f_within]
        ['above | 'below]
        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.



## 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 Constraint 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

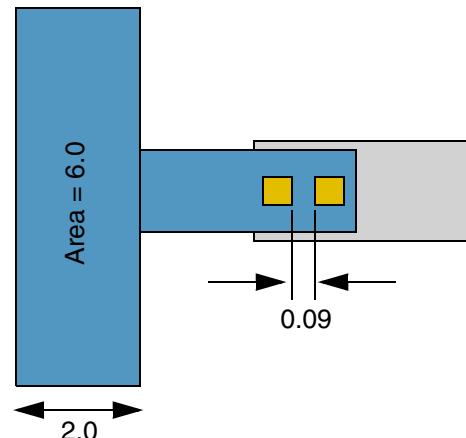
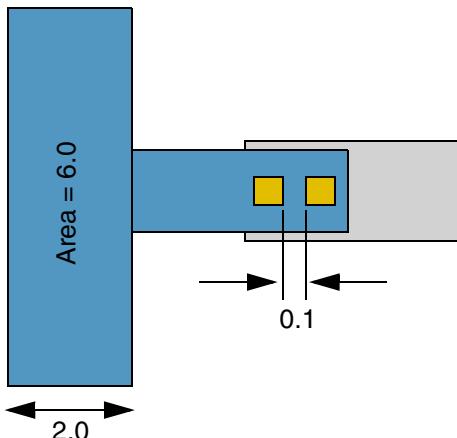
- Example 1: minProtrusionNumCut with width, area, and distanceWithin
  - Example 2: minProtrusionNumCut with distance, width, area, length, and distanceWithin

### **Example 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 "Vial"
        'width 1.0
        'area 6.0
        'distanceWithin 0.1
        2
    )
);spacings
```



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
```

## **Virtuoso Technology Data Constraint Reference**

### NumCut Constraints

---

---

## Overlap Constraints

---

This chapter includes the following constraints:

- [minDirectionalOverlap \(Advanced Nodes Only\)](#)
- [minInsideCornerOverlap](#)
- [minOverlapDistance](#)
- [minWireOverlap \(Advanced Nodes Only\)](#)

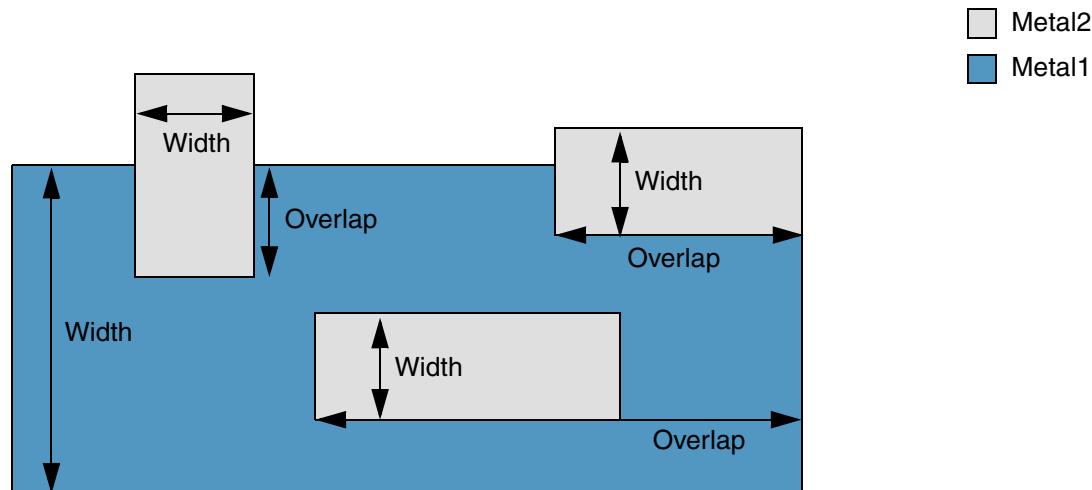
## minDirectionalOverlap (Advanced Nodes Only)

```
orderedSpacings(
    ( minDirectionalOverlap tx_layer1 tx_layer2
        ['horizontalOverlap | 'verticalOverlap]
        ['widthRanges g_ranges]
        ['otherWidthRanges g_ranges]
        ['sameNetOnly]
        f_value
    )
) ;orderedSpacings
```

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 Constraint 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

'horizontalOverlap   'verticalOverlap	The direction in which the overlap is measured. By default, the constraint applies to both vertical and horizontal overlaps. Type: Boolean
'widthRanges <i>g_ranges</i>	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.
'otherWidthRanges <i>g_ranges</i>	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.
'sameNetOnly	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 Constraint Reference

### Overlap Constraints

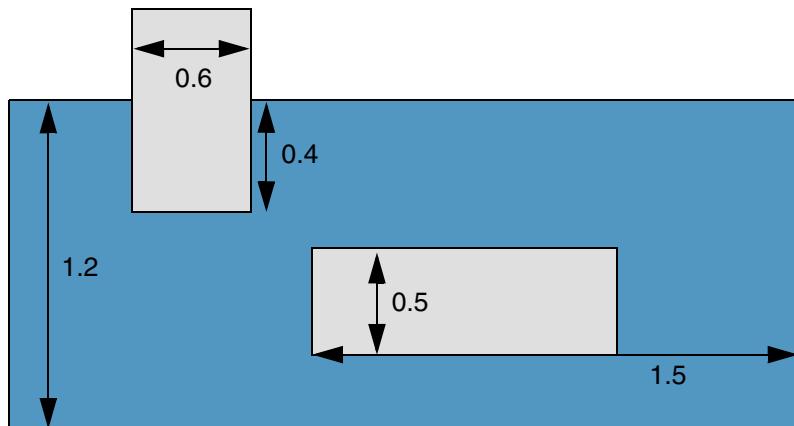
#### Example

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



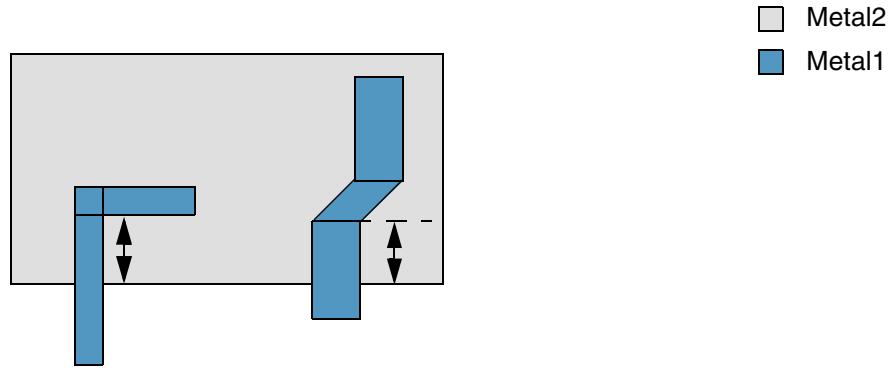
FAIL. The widths of the Metal1 and Metal2 shapes fall in the specified ranges. However, the overlap between the top Metal2 shape and the Metal1 shape is only 0.4 (<0.5).

## **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**

*tx\_layer1*      The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

*f\_space*      The overlap must be greater than or equal to this value.

### **Parameters**

None

## Virtuoso Technology Data Constraint 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 Constraint 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 (Advanced Nodes Only)

```
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
```

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

*tx\_layer1*

The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

*f\_overlap*

The overlap between the two wires must be greater than or equal to this value.

"width" nil nil

This identifies the index for *table*.

## Virtuoso Technology Data Constraint 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.
- 'sameWidth: The two wires must have the same width and must overlap exactly.

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 Constraint Reference

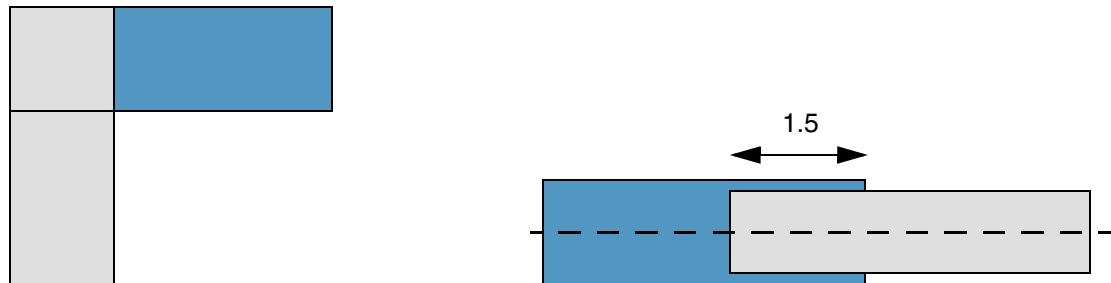
### Overlap Constraints

#### ***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 Constraint Reference

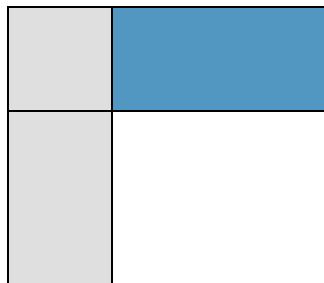
### Overlap Constraints

#### Example 2: *minWireOverlap* with *sameWidth*

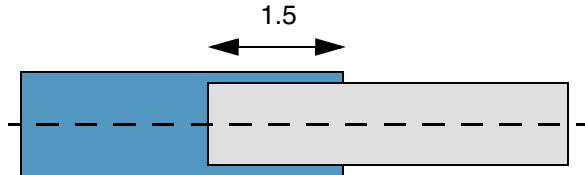
Wires on Metal1 and Metal2 must satisfy a minimum overlap value of 1.5. Additionally, they must have the same width and overlap exactly.

```
orderedSpacings(  
    ( minWireOverlap "Metal1" "Metal2"  
        'sameWidth  
        1.5  
    )  
) ;orderedSpacings
```

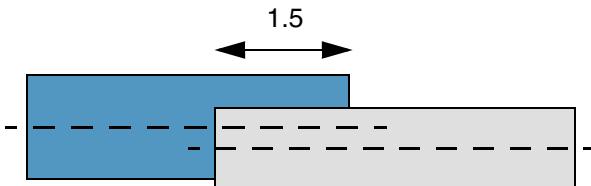
 Metal2  
 Metal1



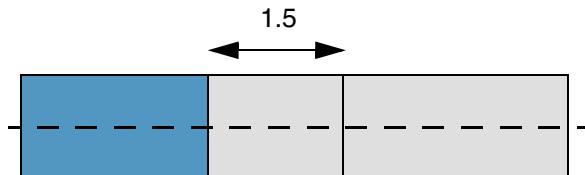
a) The constraint does not apply because the wires are perpendicular to each other. A violation would occur if '*'sameDir*' was specified.



b) FAIL. The wires do not have the same width.



c) FAIL. Wires must overlap exactly.



d) PASS. Wires have the same width and the overlap value is 1.5.

---

## Placement and Alignment Constraints

---

This chapter includes the following constraints:

- [allowedWidthSpacingPatternGroups \(ICADV12.3 Only\)](#)
- [allowedWidthSpacingPatterns \(ICADV12.3 Only\)](#)
- [defaultActiveWidthSpacingPattern \(ICADV 12.3 Only\)](#)
- [horizontalOffset](#)
- [horizontalPitch](#)
- [keepPRBoundarySharedEdges](#)
- [keepSharedEdges](#)
- [snapGridHorizontal](#)
- [snapGridVertical](#)
- [verticalOffset](#)
- [verticalPitch](#)

## **allowedWidthSpacingPatternGroups (ICADV12.3 Only)**

```
spacings(
  ( allowedWidthSpacingPatternGroups
    'snapPatternDef t_spDefName
    (l_patternNames)
  )
) ;spacings
```

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 (ICADV12.3 Only)**

```
spacings(
  ( allowedWidthSpacingPatterns
    'snapPatternDef t_snapPatternDefName
    (l_patternNames)
  )
) ;spacings
```

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 (ICADV 12.3 Only)**

```
spacings(
  ( defaultActiveWidthSpacingPattern
    'snapPatternDef t_spdefname
    t_patternName
  )
) ;spacings
```

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**

*f\_offset*      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**

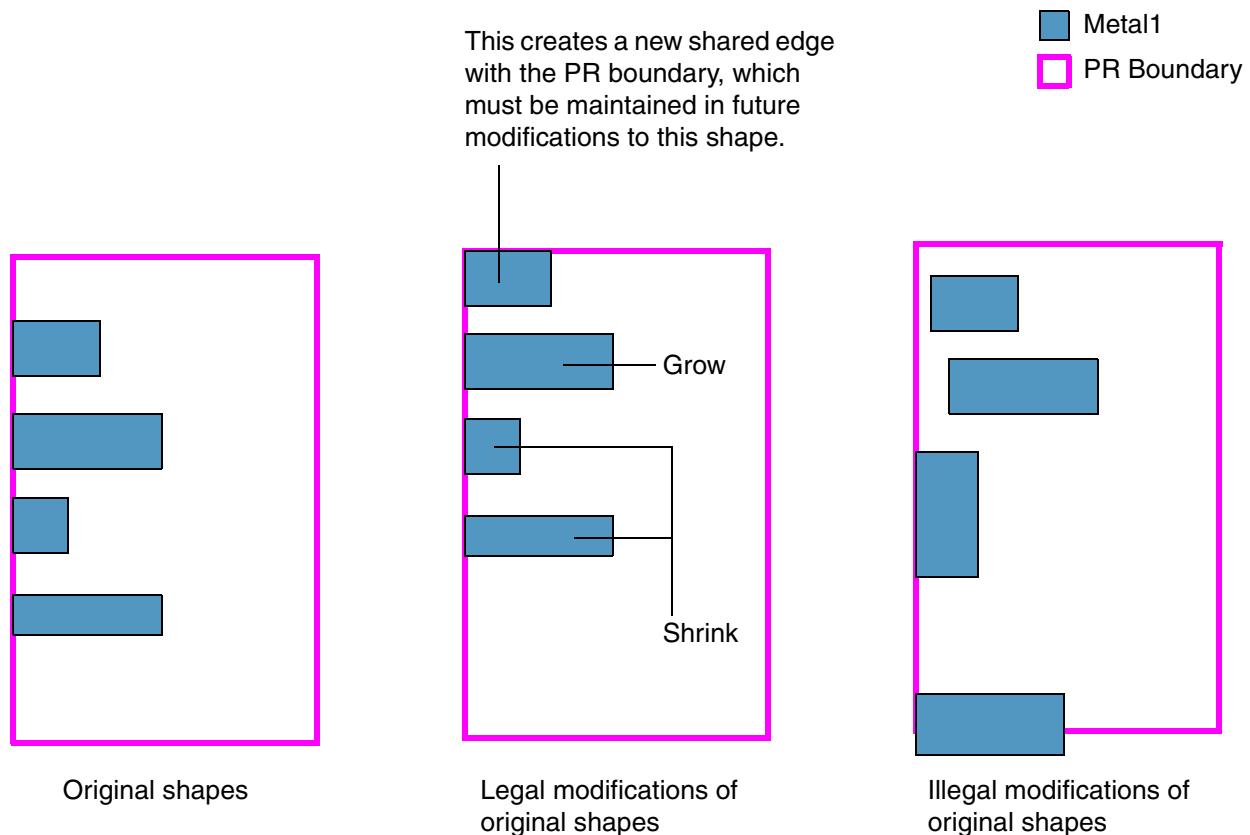
The minimum spacing required between vertical placement grid lines is 1.0.

```
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.



## 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</i>   <i>nil</i>	If <i>t</i> , the shared edges with the PR boundary must be maintained; if <i>nil</i> , the shared edges with the PR boundary need not be maintained.

## Parameters

None

## Example

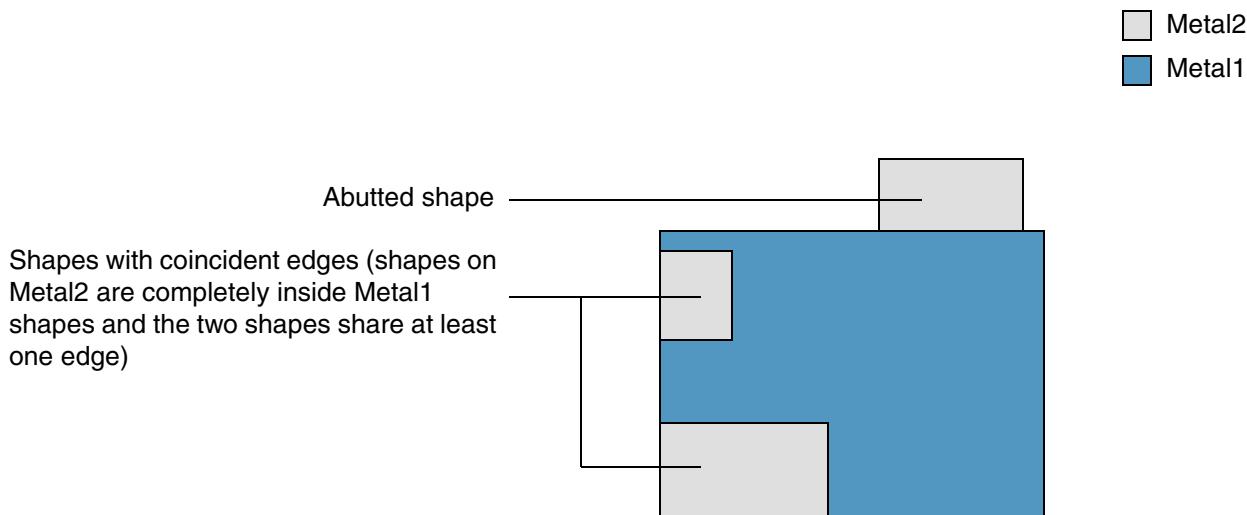
Edges of Metal1 shapes coincident with the PR boundary must remain coincident when the shapes are modified.

```
spacings(
  ( keepPRBoundarySharedEdges "Metal1"
    t
  )
) ;spacings
```

## keepSharedEdges

```
spacings(
  ( keepSharedEdges tx_layer1 tx_layer2
    { t | nil }
    [ 'butOnly | '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</i>   <i>nil</i>	If <i>t</i> , the shared edges must be maintained; if <i>nil</i> , the shared edges need not be maintained.

## 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
```

## **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 )  
)
```

Sets the minimum offset from the origin for the horizontal placement grid.

### **Parameters**

*g\_offset*      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
```

---

## Routing Constraints

---

This chapter includes the following constraints:

- [clusterDistance](#)
- [excludeLPPs](#)
- [horizontalOffset](#)
- [horizontalPitch \(One layer\)](#)
- [ladderFrequency](#)
- [ladderOffset](#)
- [ladderStyle](#)
- [layerMaskShiftAllowed \(ICADV12.3 Only\)](#)
- [leftDiagOffset](#)
- [leftDiagPitch](#)
- [maxRoutingDistance](#)
- [numStrands](#)
- [orthogonalSnappingLayer \(ICADV12.3 Only\)](#)
- [orthogonalWSPGrid \(ICADV12.3 Only\)](#)
- [rectangularGapMinSpacing \(ICADV12.3 Only\)](#)
- [rightDiagOffset](#)
- [rightDiagPitch](#)
- [routingDirections](#)
- [strandSpacing](#)
- [strandWidth](#)

## Virtuoso Technology Data Constraint Reference

### Routing Constraints

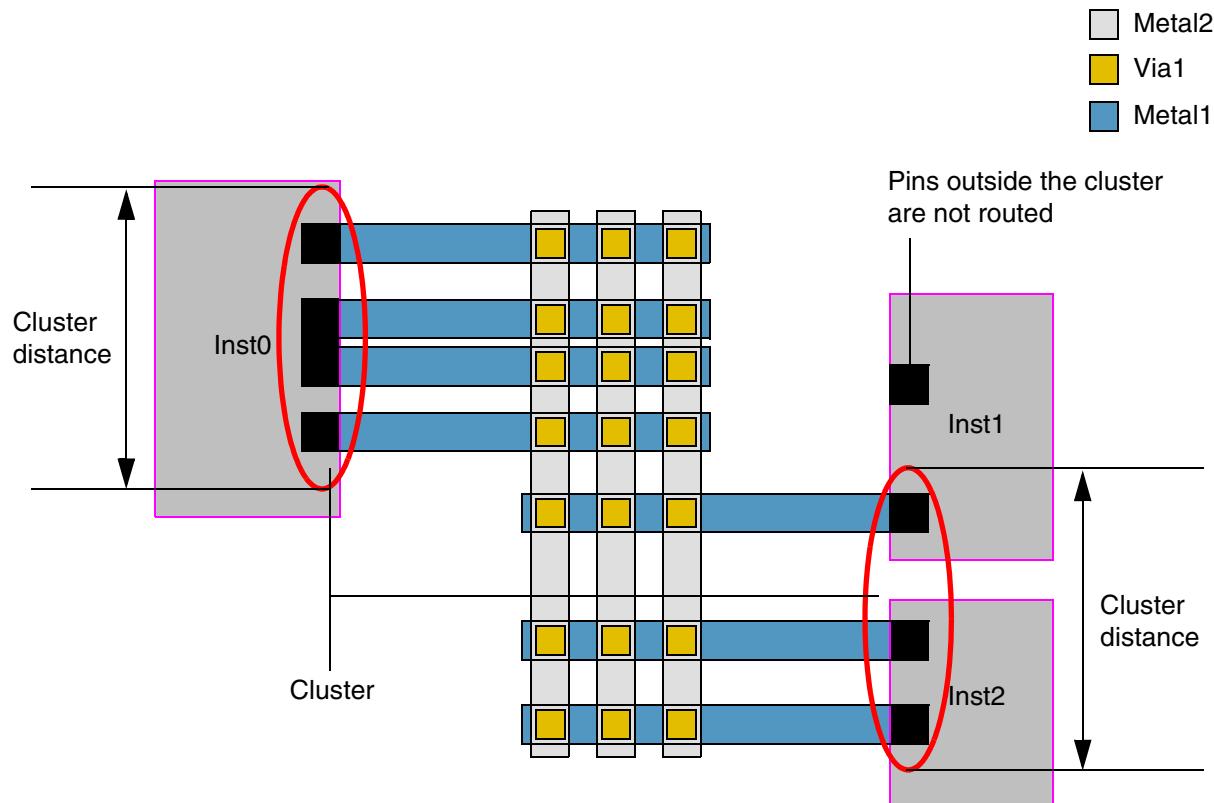
---

- [trackPattern](#)
- [taperHalo](#)
- [validLayers](#)
- [validPurposes](#)
- [validVias](#)
- [verticalOffset](#)
- [verticalPitch \(One layer\)](#)

## clusterDistance

```
spacings(
    ( clusterDistance f_clusterDistance )
) ;spacings
```

Applies to a multi-pin net in which pins spaced at a distance less than or equal to *clusterDistance* apart are clustered together.



## 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**

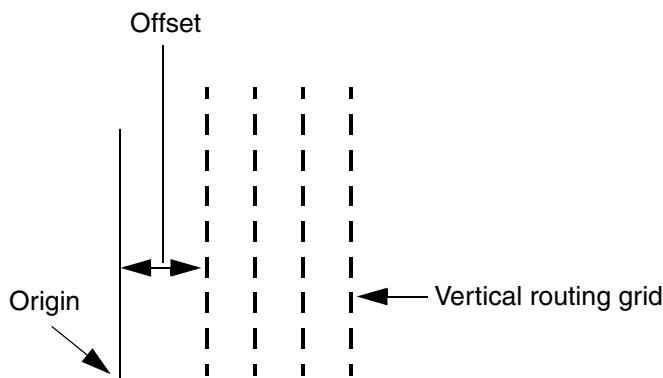
For layers M0, M1, M2, and M3, all layer-purpose pairs with purpose "drawing", "vdd", "vss", "test1", and "test2" are included in connectivity extraction, except layer-purpose pairs "M0, test1" and "M1, test2".

```
("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

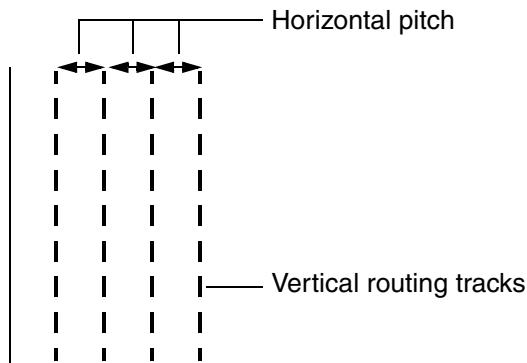
The horizontal offset must be 1.2 for Metal3 and 1.5 for the rest of the layers.

```
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 ' <i>firstLastPitch</i> ' is specified, this distance applies only to the tracks between the first and last routing tracks.

### Parameters

<i>'firstLastPitch f_firstLastPitch'</i>	The distance of the first and the last routing track from the cell row boundary.
--	--

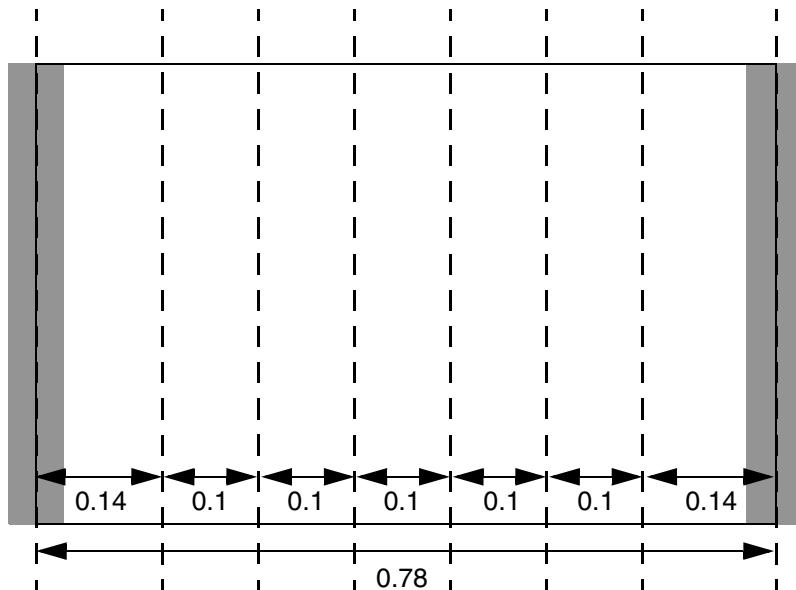
## Virtuoso Technology Data Constraint Reference

### 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**

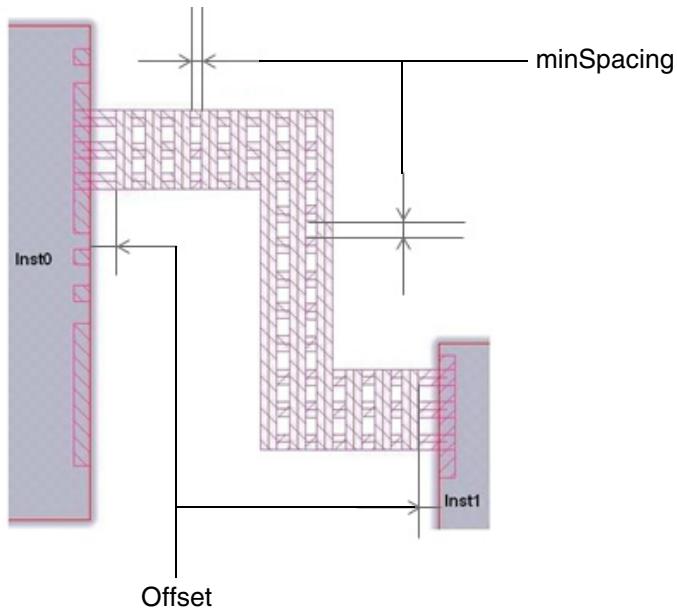
*f\_frequency*      The laddering percentage.

### **Parameters**

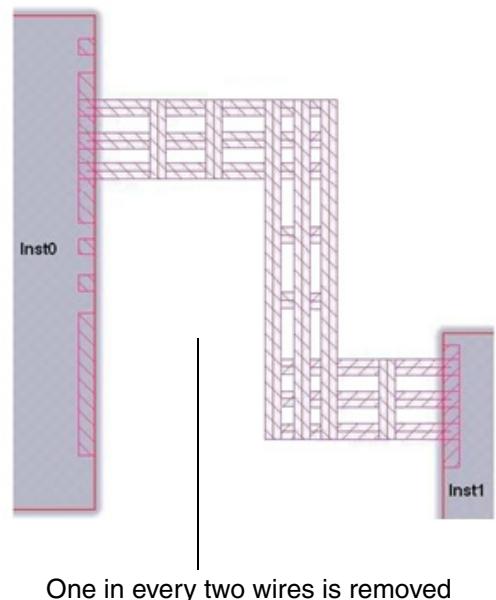
None

### **Example**

Frequency = 100%



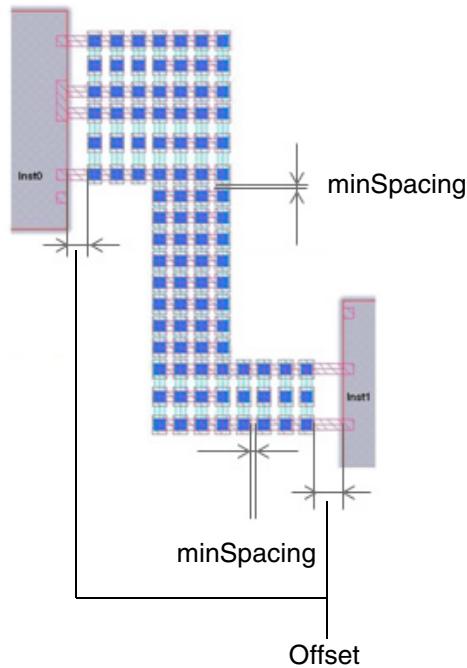
Frequency = 50%



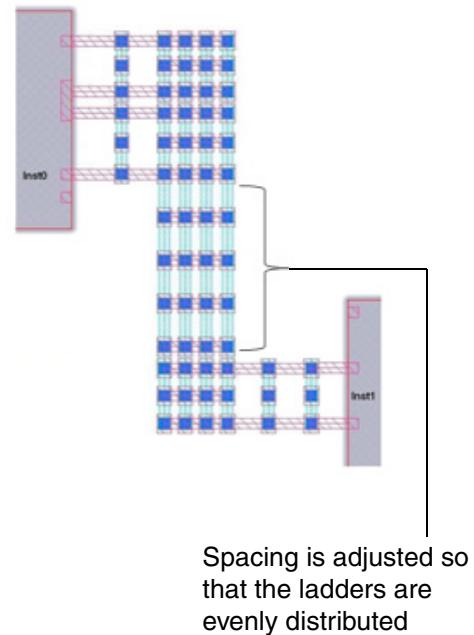
# Virtuoso Technology Data Constraint Reference

## 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**

*tx\_layer*      The layer on which the constraint is applied.

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

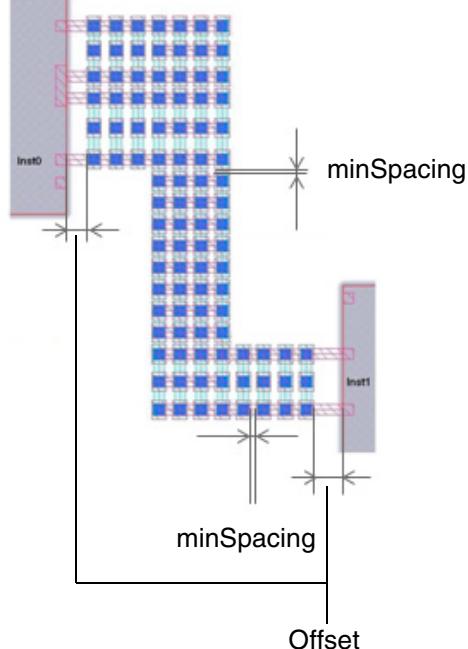
*f\_offset*      The offset.

### **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.

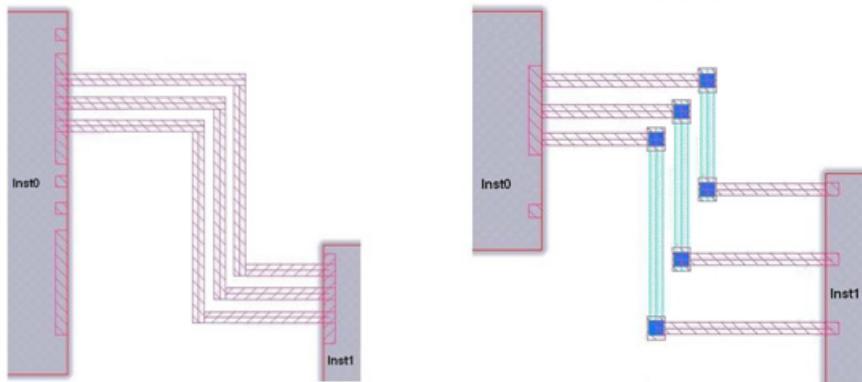
- **'sameLayerLadder:** The wire strands and the associated perpendicular wires are on the same layer.
- **'layerAboveLadder:** The associated perpendicular wires are on the layer above the wire strands.
- **'layerBelowLadder:** The associated perpendicular wires are on the layer below the wire strands.

# Virtuoso Technology Data Constraint Reference

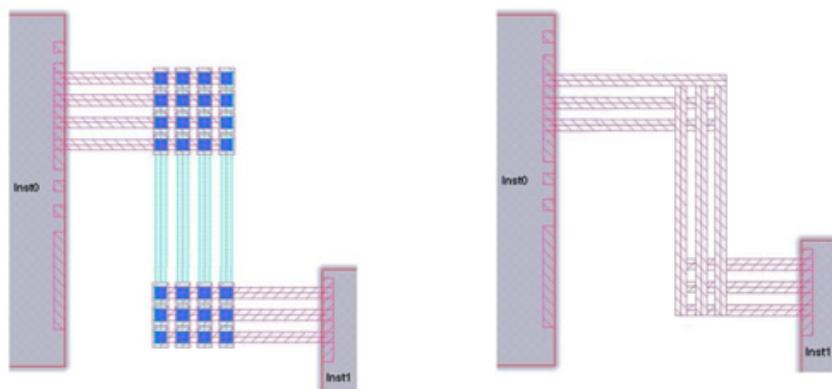
## Routing Constraints

### Examples

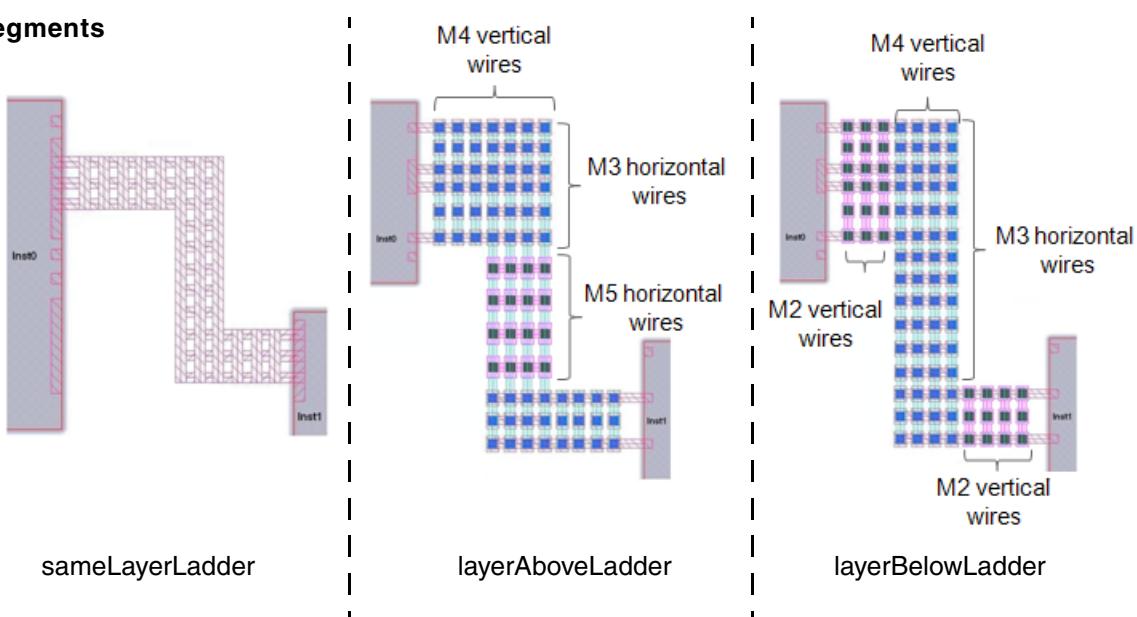
#### noLadderStyle



#### atTurnOnly



#### allSegments



## **layerMaskShiftAllowed (ICADV12.3 Only)**

```
spacings(  
    ( layerMaskShiftAllowed (l_layerNames) )  
) ;spacings
```

Specifies the layers on which masks can be shifted.

### **Values**

*l\_layerNames*      The layers on which the constraint is applied.

Type: List of layer names

### **Example**

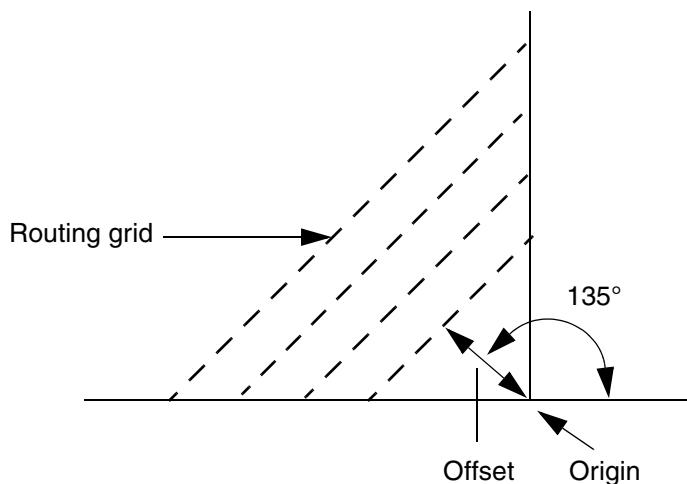
Mask shifting is allowed only on layers Metal1 through Metal4 and Via1.

```
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

## **Example**

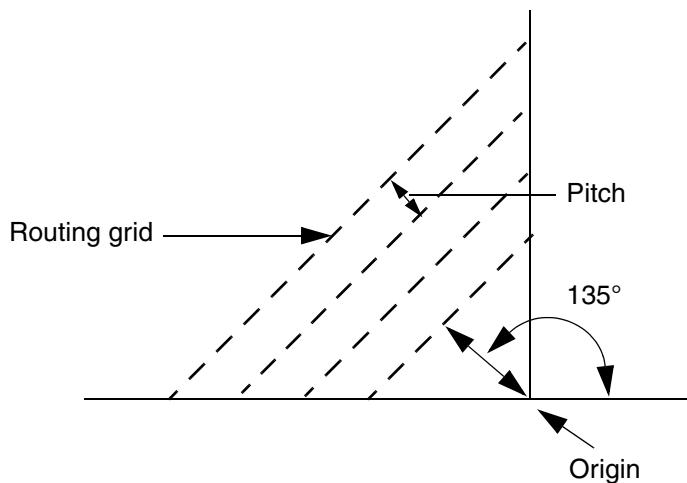
The left diagonal offset must be 1.2 for Metal3 and 1.4 for the rest of the layers.

```
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**

*tx\_layer*      The layer on which the constraint is applied.

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

*f\_pitch*      The spacing between the routing grid lines.

## **Parameters**

None

## Virtuoso Technology Data Constraint 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**

*tx\_layer*                   The layer on which the constraint is applied.

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

*f\_distance*               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**

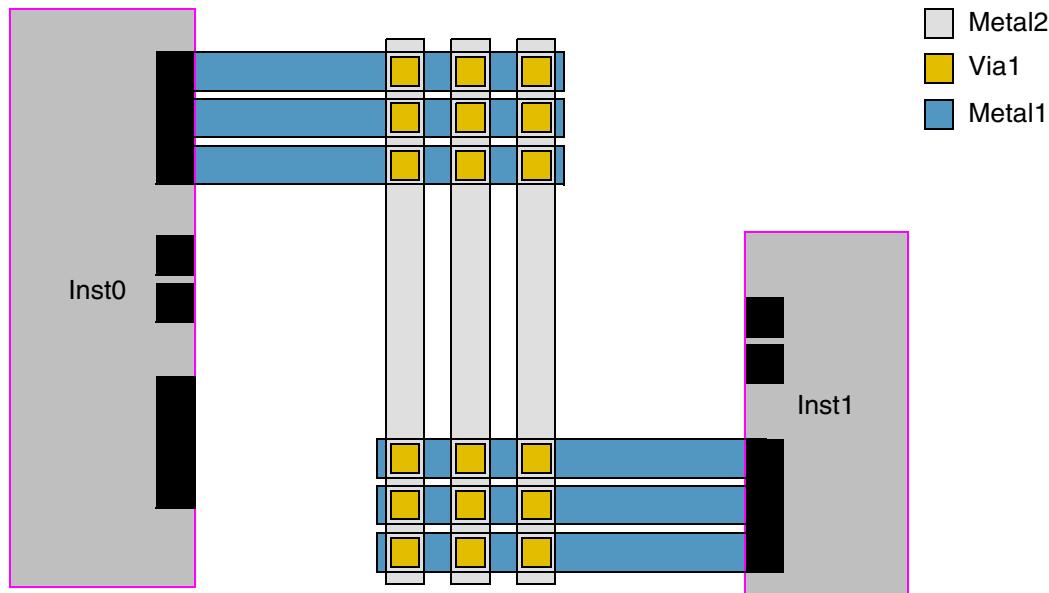
- [Example 1: Fat-to-fat pins](#)
- [Example 2: Fat-to-many pins](#)
- [Example 3: Many-to-many pins](#)

# Virtuoso Technology Data Constraint Reference

## Routing Constraints

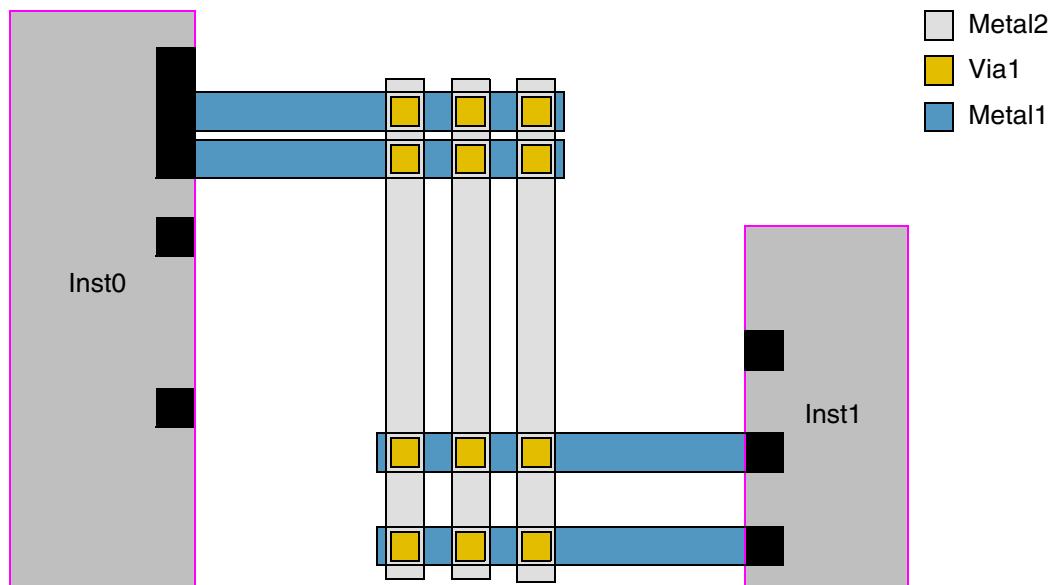
### **Example 1: Fat-to-fat pins**

The pin is split to accommodate the number of ( $n=3$ ) strands.



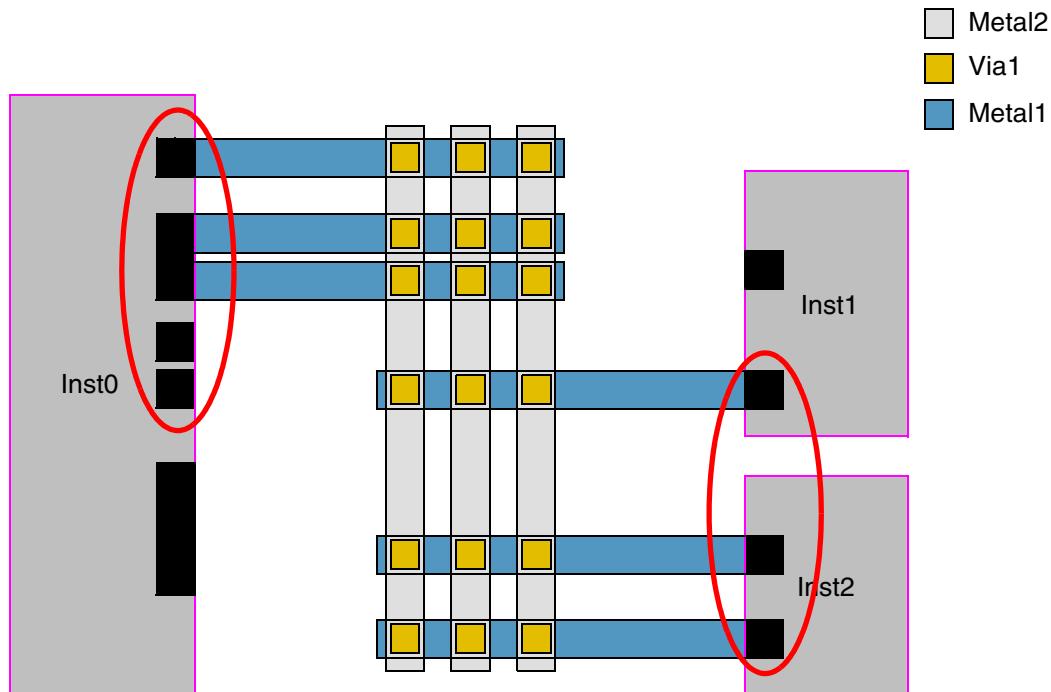
### **Example 2: Fat-to-many pins**

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.



***Example 3: Many-to-many pins***

The number of strands ( $n=3$ ) is distributed among the pins in the cluster.



## **orthogonalSnappingLayer (ICADV12.3 Only)**

```
spacingTables(
  ( orthogonalSnappingLayer tx_layer1 tx_layer2
    (( "width" nil nil )
     ['upperLineEndStoppingPoints g_upperTable]
     ['mask1 | 'mask2 | 'mask3]
     ['otherMask1 | 'otherMask2 | 'otherMask3]
     l_defaultRanges
    )
    g_rangeTable
  )
) ;spacingTables
```

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**

*tx\_layer1*                   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)

*tx\_layer1*                   The second layer on which the constraint is applied. The line-end edges on *layer1* 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 *rangeTable*.

*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 actual 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.

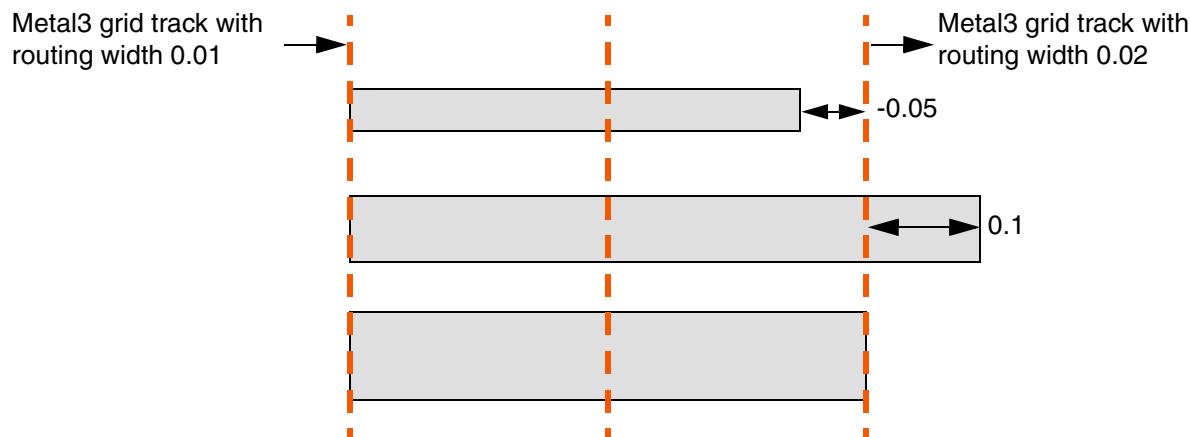
## Example

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 (ICADV12.3 Only)**

```

spacings(
    ( orthogonalWSPGrid tx_layer
        [ 'upperLineEndWSSPDefName t_upperLineEndWsspDefName]
        [ 'mask1 | 'mask2 | 'mask3]
        t_lineEndWSSPDefName
    )
) ; spacings

```

Specifies that the line-ends on the given layer must coincide with the defaultActive WSP grid of the specified widthSpacingSnapPatternDef.

If 'upperLineEndWSPDefName' is not specified, the defaultActive WSP grid applies to both lower and upper line-ends.

Optionally, the constraint applies only to the shapes with the specified mask. The color of the tracks in the WSP grid referenced by the constraint is ignored.

## Values

*tx\_layer* The layer on which the constraint is applied.

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

*t\_lineEndWSSPDefName*

The widthSpacingSnapPatternDef name.

## Parameters

'upperLineEndWSSPDefName t\_upperLineEndWsspDefName

The upper line-ends must coincide with this WSP grid, and the lower line-ends must coincide with the `defaultActive` WSP grid.

Otherwise, the `defaultActive` WSP grid applies to both upper and lower line-ends.

'mask1 | 'mask2 | 'mask3

The constraint applies only to the shapes on this mask.

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 (ICADV12.3 Only)**

```

spacingTables(
    ( rectangularGapMinSpacing tx_layer
        (( "track" nil nil )
            'exactWidth f_exactWidth 'maxLength f_maxLength
            [ 'endToEndSpacing f_endToEndSpacing]
        )
        (g_table)
    )
)

; spacingTables

```

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

*tx\_layer* 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 *table*.

*g\_table* 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.

## 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 Constraint Reference

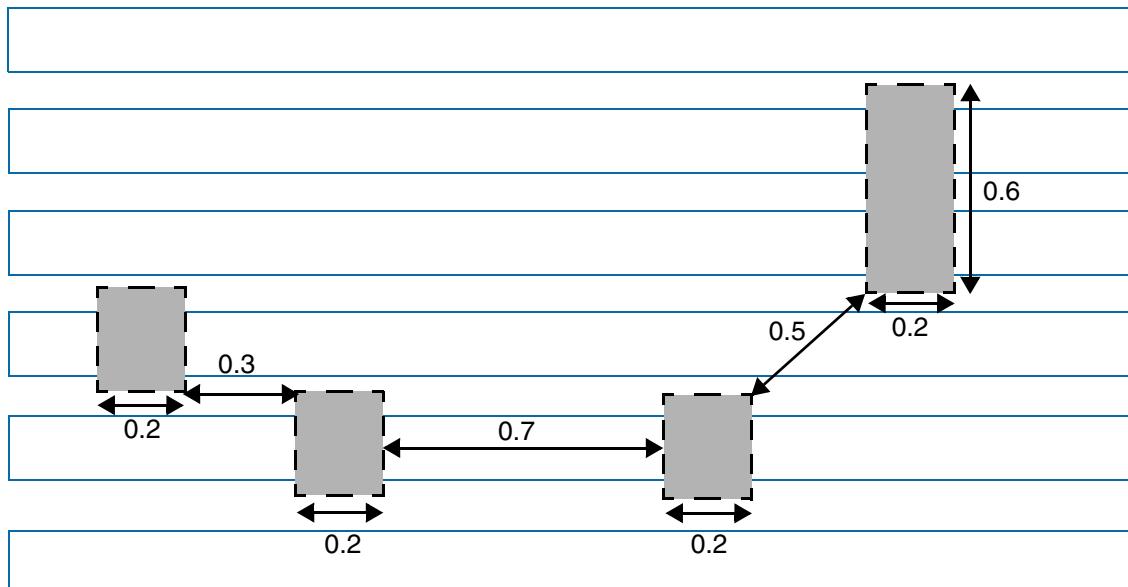
### Routing Constraints

---

```

spacingTables(
    ( rectangularGapMinSpacing "Metall1"
        (( "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 Constraint Reference

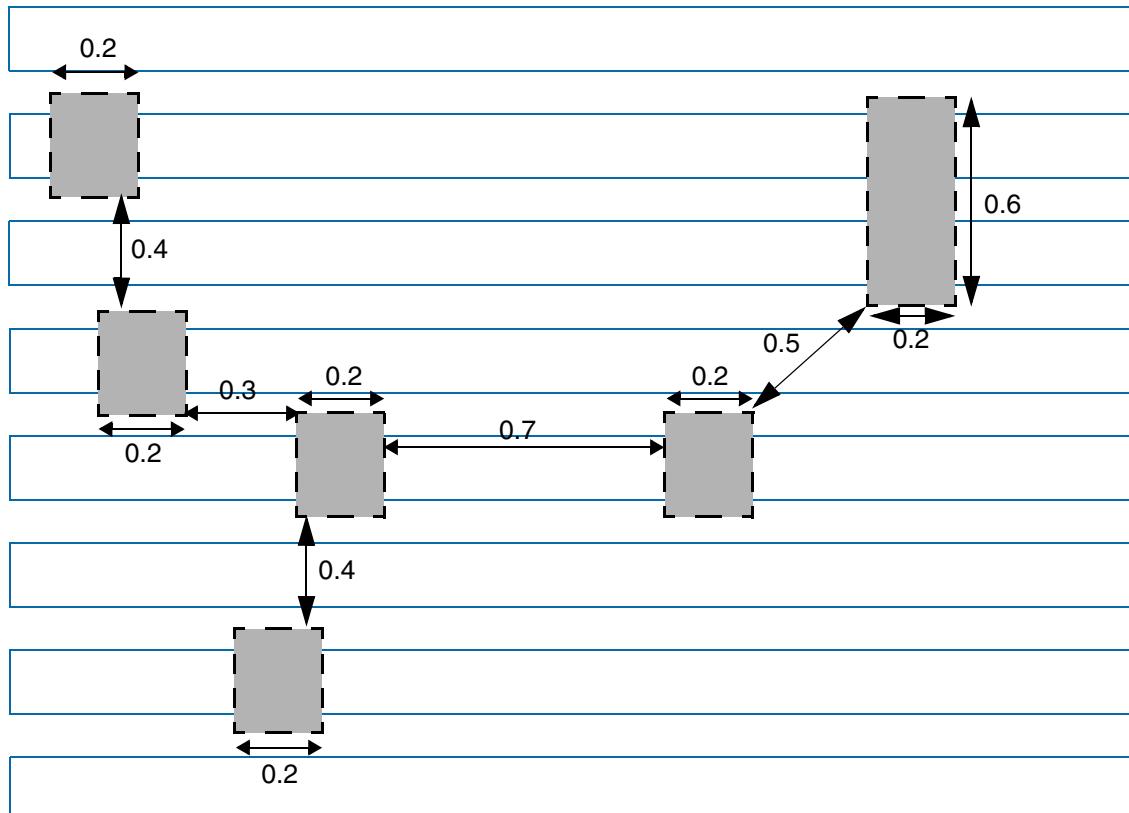
### Routing Constraints

---

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
```

 Metal1  
 Gap

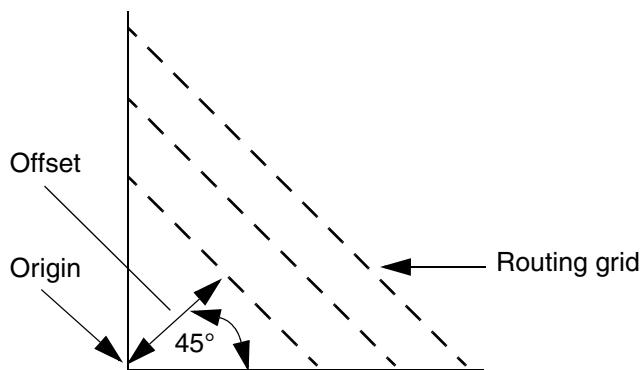


PASS. The width of the gaps is 0.2 and the length is less than or equal to 0.6. The spacing between the two middle gaps on the same track is 0.7; spacing between the leftmost gaps on adjacent tracks is 0.3; spacing for the two rightmost gaps that are two tracks apart is 0.5; and the end-to-end spacing 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

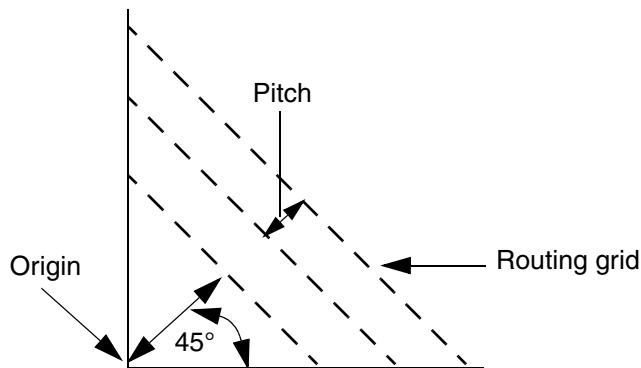
The right diagonal offset must be 1.2 for Metal3 and 1.4 for the rest of the layers.

```
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**

*tx\_layer*      The layer on which the constraint is applied.

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

*f\_pitch*      The spacing between the routing grid lines.

### **Parameters**

None

### **Example**

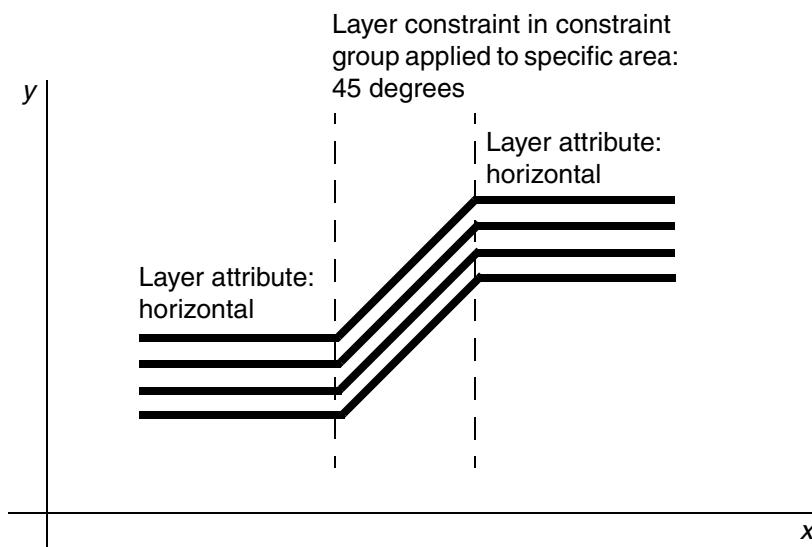
The right diagonal pitch must be 1.0 for Metal3 and 1.2 for the rest of the layers.

```
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**

<code>tx_layer</code>	The layer on which the constraint is applied.  Type: String (layer and purpose names) or Integer (layer number)
<code>t_direction</code>	The routing direction for the layer inside any area to which the constraint group is applied. Valid values include <code>vertical</code> , <code>horizontal</code> , <code>leftDiag</code> , <code>rightDiag</code> , and <code>none</code> . <ul style="list-style-type: none"><li>■ <code>leftDiag = 135 degrees</code></li><li>■ <code>rightDiag = 45 degrees</code></li></ul>

## Parameters

None

## Example

Layer attributes are defined as follows:

```
layerRules(
    routingDirections(
        ("Metal1" "horizontal")
        ("Metal2" "vertical")
        ("Metal3" "rightDiag")
        ("Diff" "none")
        ("Via1" "notApplicable")
    ) ;routingDirections
) ;layerRules
```

The routingDirections constraint group is defined as follows:

```
constraintGroups(
    ("myRoutingLayerOverrides"
        routingDirections(
            ("Metal1" "rightDiag")
            ("Metal2" "horizontal")
            ("Metal3" "none")
        ) ;routingDirections
    ) ;myRoutingLayerOverrides
) ;constraintGroups
```

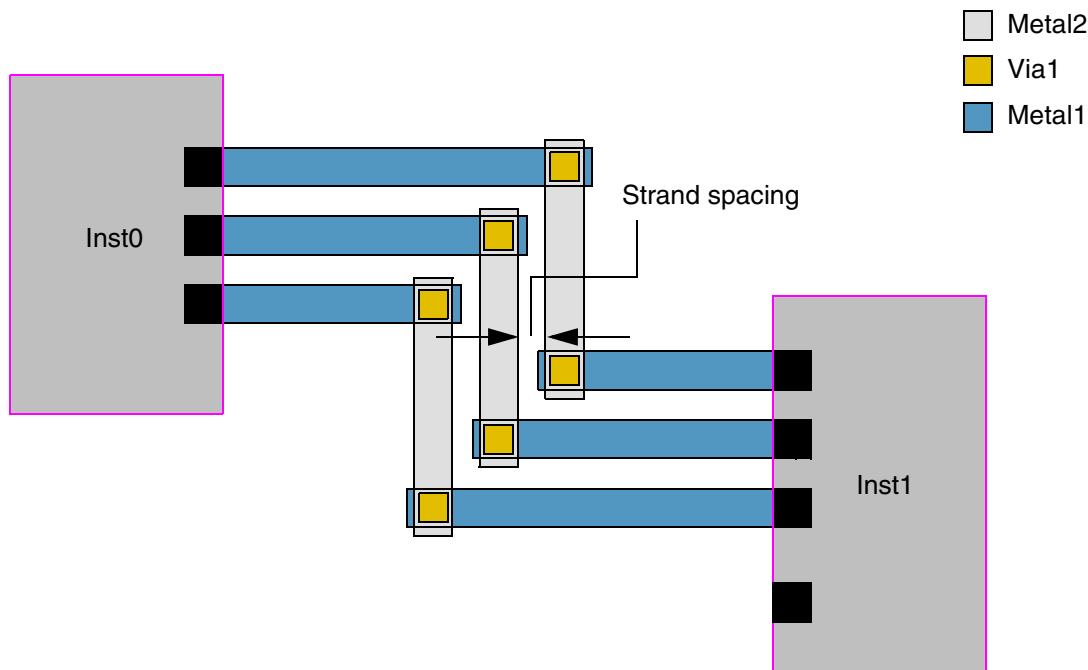
Applying the myRoutingLayerOverrides constraint group to a bounded area sets the following routing directions within that area:

- 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

## strandSpacing

```
spacings(  
    ( strandSpacing tx_layer f_spacing )  
) ;spacings
```

Specifies the spacing between individual wire strands.



## 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 wire strands.

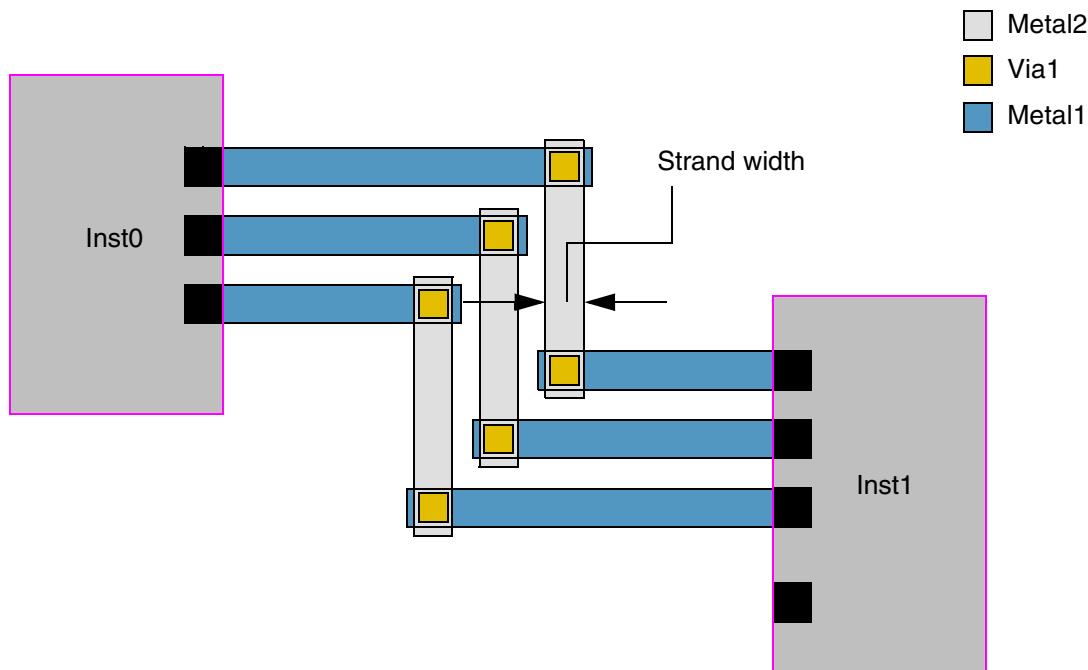
## Parameters

None

## strandWidth

```
spacings(  
  ( strandWidth tx_layer f_width )  
) ;spacings
```

Specifies the width of an individual wire strand.



## 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 an individual wire strand.

## Parameters

None

## **trackPattern**

```
spacings(  
  ( trackPattern tx_layer t_trackPatternName )  
) ;spacings
```

Specifies track patterns for track-based routing.

### **Values**

*tx\_layer*      The layer on which the constraint is applied.

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

*t\_trackPatternName*

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**

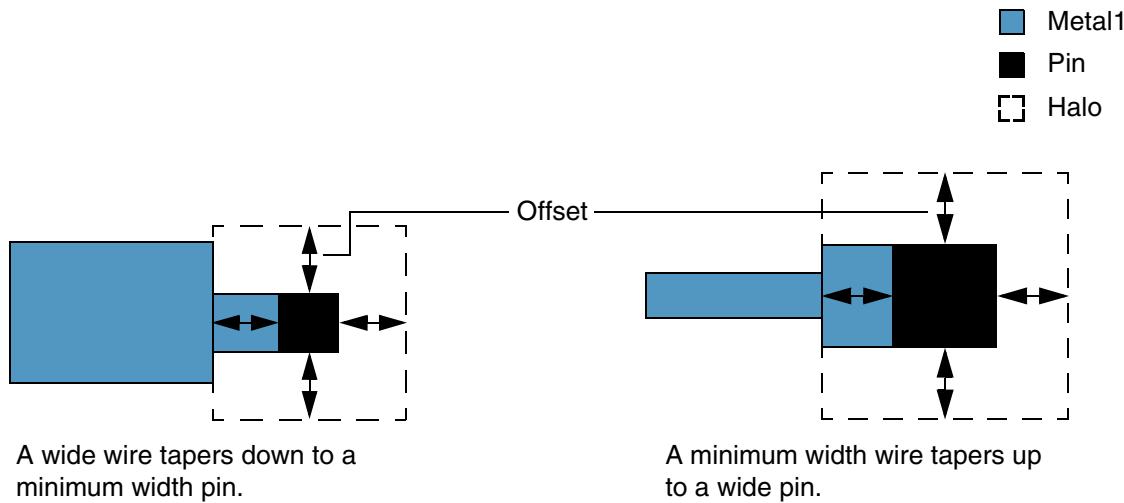
Adding the following constraint to a constraint group associated with a net restricts the routing of that net on Metal1 to tracks with track pattern "tp1".

```
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**

*tx\_layer1*                          The layer on which the constraint is applied.

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

*tx\_layer1 tx\_purpose1*

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

*t\_purpose* Purposes to be included or excluded.

Type: A list of valid purposes, each enclosed in double quotes and separated by a space.

## Parameters

'include | 'exclude

**Include or exclude purposes.**

- 'include: The listed purposes are included in the analysis by the target application.
  - 'exclude: 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

'viaLayerDirection A pair of direction values corresponding to each entry in *table*. The first direction value applies to *layer1* and the second direction applies *layer2* in the via definition.

The table (*dirTable*) has the following format:

*x\_index* ((*t\_direction1 t\_direction2*)  
(*t\_direction1 t\_direction2*) ...)

where,

- *index* is a positive integer value starting at 0.
- *direction1* and *direction2* can be horizontal or vertical.

Type: A 1-D table specifying integer and string values.

*l\_default* 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 Constraint Reference

### Routing Constraints

---

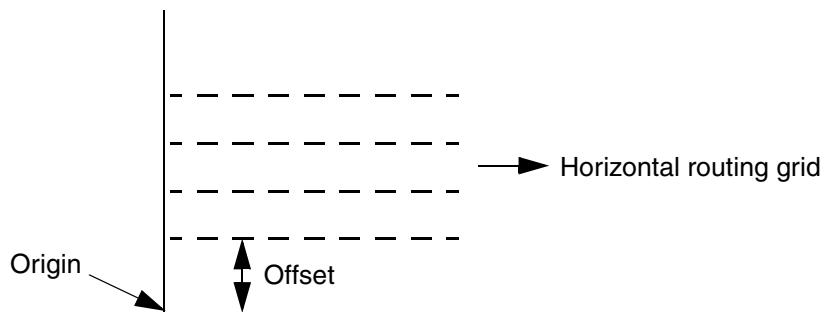
#### ***Example 3: validVias with viaLayerDirection***

```
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**

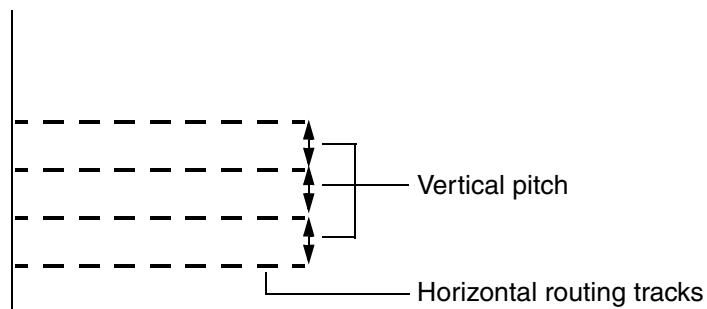
The vertical offset must be 1.2 for Metal3 and 1.4 for the rest of the layers.

```
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**

*tx\_layer*      The layer on which the constraint is applied.

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

*f\_pitch*      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.

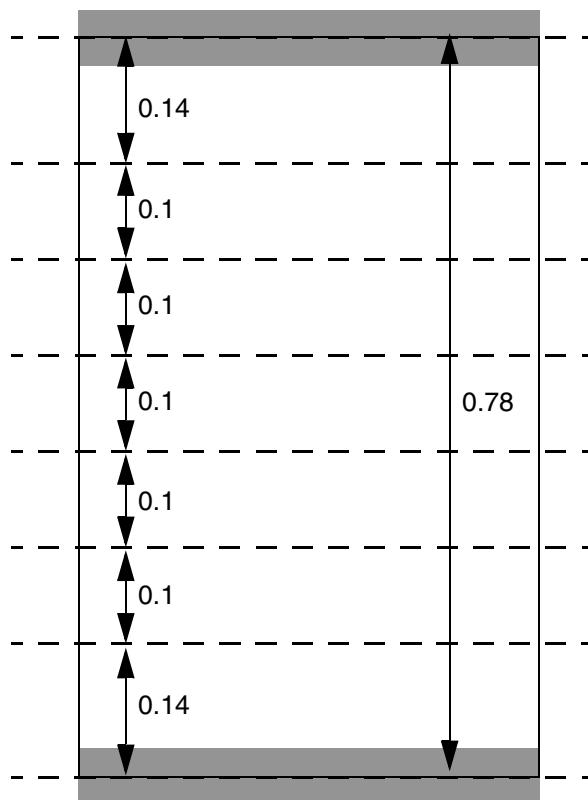
## Virtuoso Technology Data Constraint Reference

### 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 ( $2 \times 0.14 + 5 \times 0.1$ ).

## **Virtuoso Technology Data Constraint Reference**

### Routing Constraints

---

---

## Spacing Constraints (One Layer)

---

This chapter includes the following constraints:

- [allowedBetweenNeighborWidthRanges \(ICADV12.3 Only\)](#)
- [allowedSpacingRanges \(One layer\)](#)
- [cornerFillSpacing](#)
- [endOfLineKeepout](#)
- [forbiddenEdgePitchRange \(Advanced Nodes Only\)](#)
- [forbiddenProximitySpacing \(Advanced Nodes Only\)](#)
- [gateSpacingRanges](#)
- [maxFilling](#)
- [maxTapSpacing](#)
- [minAdjacentFourViaSpacing \(ICADV12.3 Only\)](#)
- [minCenterToCenterSpacing](#)
- [minClusterSpacing \(One layer\) \(Advanced Nodes Only\)](#)
- [minConcaveCornerSpacing \(ICADV12.3 Only\)](#)
- [minConvexCornerSpacing](#)
- [minCornerSpacing \(One layer\) \(Advanced Nodes Only\)](#)
- [minCornerToCornerDistance](#)
- [minDiagonalSpacing](#)
- [minDiffNetSpacing](#)
- [minEdgeLengthSpacing \(Advanced Nodes Only\)](#)
- [minEndOfLineExtensionSpacing](#)

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

- [minEndOfLinePerpSpacing](#)
- [minEndOfLineSpacing](#)
- [minEndOfLineToConcaveCornerSpacing](#)
- [minEndOfLineToNotchSpacing \(Advanced Nodes Only\)](#)
- [minEndOfNotchSpacing](#)
- [minExtensionSpacing](#)
- [minFillToFillSpacing](#)
- [minFillToShapeSpacing](#)
- [minFiveWiresEndOfLineSpacing \(ICADV12.3 Only\)](#)
- [minHoleWidth](#)
- [minInfluenceSpacing](#)
- [minJointCornerSpacing](#)
- [minLargeNeighborViaArrayCutSpacing](#)
- [minNotchSpacing](#)
- [minNotchSpanSpacing](#)
- [minOppSpanSpacing](#)
- [minOrthogonalSpacing \(ICADV12.3 Only\)](#)
- [minOuterVertexSpacing](#)
- [minParallelSpanSpacing](#)
- [minPRBoundaryExteriorHalo](#)
- [minPRBoundaryInteriorHalo](#)
- [minPrlTwoSidesSpacing \(ICADV12.3 Only\)](#)
- [minProtrusionSpacing](#)
- [minSameNetSpacing \(One layer\)](#)
- [minSideSpacing \(One layer\) \(Advanced Nodes Only\)](#)
- [minSpacing \(One layer\)](#)
- [minSpanLengthSpacing \(Advanced Nodes Only\)](#)

## **Virtuoso Technology Data Constraint Reference**

### Spacing Constraints (One Layer)

---

- [minStubInfluenceSpacing](#)
- [minVoltageSpacing \(One layer\)](#)
- [oneSideSpacing \(ICADV12.3 Only\)](#)
- [pgViaTrack \(Advanced Nodes Only\)](#)
- [shapeRequiredBetweenSpacing \(Advanced Nodes Only\)](#)
- [snapPatternDefOffset \(ICADV12.3 Only\)](#)
- [trimMinAdjacentSpacing \(ICADV12.3 Only\)](#)
- [trimMinSpacing \(One layer\) \(ICADV12.3 Only\)](#)
- [viaKeepoutZone \(ICADV12.3 Only\)](#)

## **allowedBetweenNeighborWidthRanges (ICADV12.3 Only)**

```
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
```

Specifies the allowed width ranges for a shape, using a 2-D table indexed by the width of the neighboring shapes.

### **Values**

*tx\_layer*                   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 *table*.

*g\_table*                   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 ranges.

### **Parameters**

'horizontal | 'vertical

The spacing applies in the specified direction.

Type: Boolean

## Virtuoso Technology Data Constraint 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 exactly 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 Constraint 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 ( $\leq 0.7$ ). 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.

**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].

## **allowedSpacingRanges (One layer)**

```
spacings(
  ( allowedSpacingRanges tx_layer
    ['width f_width']
    ['horizontal | 'vertical]
    ['widthRanges (g_widthRanges)']
    ['exceptOverLayer tx_exceptOverLayer
      'overLayerPrl f_prl
      'overLayerWidthRanges (g_overLayerWidthRanges)
    ]
    ['exceptNumShapes g_exceptNumShapeRange
      'numShapeDistance f_distance
      'numShapeWidthRanges (g_numShapeWidthRanges)
      ['numShapeMaxWidth f_maxWidth]
    ]
    ['stepSize f_stepSize ['stepRange g_stepRange]]
    ['overLayer tx_overLayer]
    (g_ranges)
  )
)

) ;spacings

spacingTables(
  ( allowedSpacingRanges tx_layer
    (( "width" nil nil ["width" nil nil] )
     ['width f_width']
     ['parallelRunLengthTable]
     ['horizontal | 'vertical]
     ['widthRanges (g_widthRanges)']
     ['exceptOverLayer tx_layer
       'overLayerPrl f_prl
       'overLayerWidthRanges (g_overLayerWidthRanges)
     ]
     ['exceptNumShapes g_exceptNumShapeRange
       'numShapeDistance f_distance
       'numShapeWidthRanges (g_numShapeWidthRanges)
     ]
     [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.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

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

`tx_layer`      The layer on which the constraint is applied.

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

`g_ranges`      The allowed spacing ranges specified as follows:

(`spacingRange1` `spacingRange2` ...)

Type: Floating-point values specifying a range of spacings that are allowed.

"width" nil nil ["width" nil nil]

This identifies the index for `table`.

## Virtuoso Technology Data Constraint 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

'width *f\_width*

The constraint applies only if the width of both adjacent shapes is less than this value.

'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*)

(Advanced Nodes Only) The constraint applies only to shapes with width in this range.

Type: Floating-point values specifying a range of widths that trigger the constraint.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

```
'exceptOverLayer tx_exceptOverLayer 'overLayerPrl f_prl  
'overLayerWidthRanges (g_overLayerWidthRanges)
```

**Note:** These parameters together define an exemption that applies to two adjacent shapes.

(Advanced Nodes Only) 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.

```
'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.

(Advanced Nodes Only) 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*.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'stepSize *f\_stepSize*

(Advanced Nodes Only) The allowed spacings ( $S_{legal}$ ) are restricted to the following values:

$$S_{legal} = S_{min} + n * stepSize$$

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*

(Advanced Nodes Only) 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.

'overLayer *tx\_overLayer*

(Advanced Nodes Only) 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

*f\_default* The spacing value to be used when no table entry applies.

## 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](#)

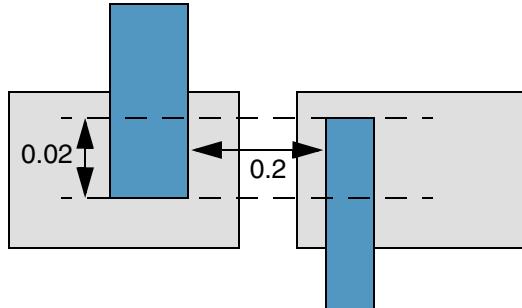
**Example 1: allowedSpacingRanges with horizontal, exceptOverLayer, overLayerPrl, and overLayerWidthRanges**

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:

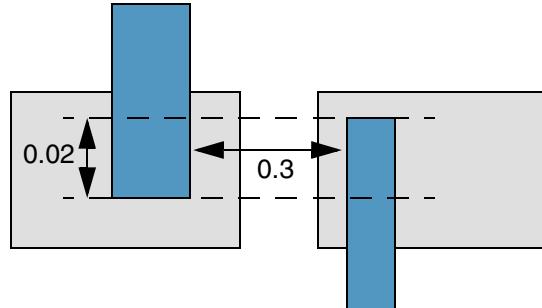
- The shapes overlap Metal2 shapes.
- The parallel run length between Metal1 shapes is less than or equal to 0.02.
- The spacing between Metal1 shapes is greater than or equal to 0.2 and less than or equal to 0.25.

```
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.

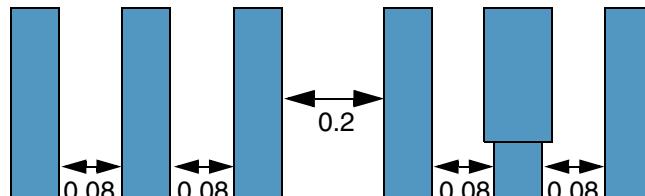
**Example 2: allowedSpacingRanges with horizontal, exceptNumShapes, numShapeDistance, and numShapeWidthRanges**

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:

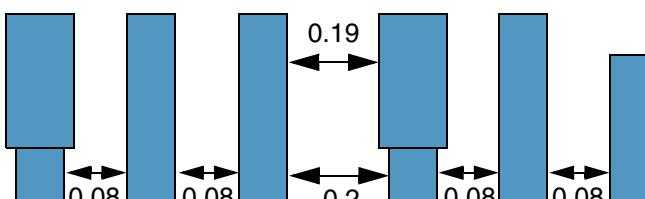
- The number of Metal1 shapes in a group is 3 or less.
- The distance between a pair of shapes in a group is less than or equal to 0.08.
- The spacing between two groups of shapes is greater than or equal to 0.15 and less than or equal to 0.2.

```
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.

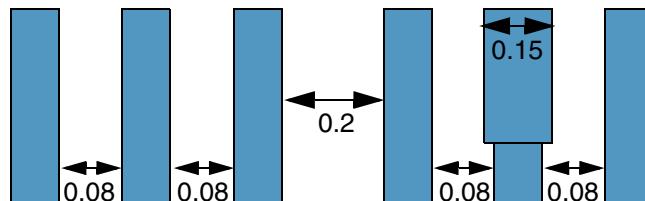
**Example 3: allowedSpacingRanges with horizontal, exceptNumShapes, numShapeDistance, numShapeWidthRanges, and numShapeMaxWidth**

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:

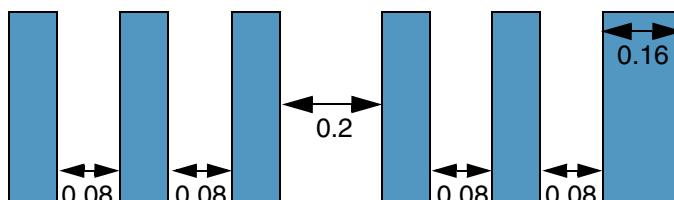
- The number of Metal1 shapes in a group is 3 or less.
- The distance between a pair of shapes in a group is less than or equal to 0.08.
- The spacing between two groups of shapes is greater than or equal to 0.15 and less than or equal to 0.2.
- The width of each shape in the group is less than or equal 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 Constraint Reference

### Spacing Constraints (One Layer)

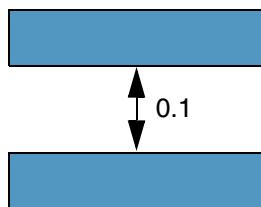
---

#### ***Example 4: allowedSpacingRanges with vertical and stepSize***

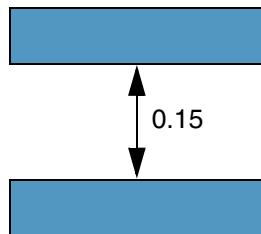
The vertical spacing between two Metal1 shapes must be greater than or equal to 0.1 and less than or equal to 0.2. A legal spacing value is the sum of the lower bound of this spacing range (0.1) and a multiple of 0.05.

```
spacings(
  ( allowedSpacingRanges "Metal1"
    'vertical
    'stepSize 0.05
    (" [0.1 0.2] ")
  )
) ;spacings
```

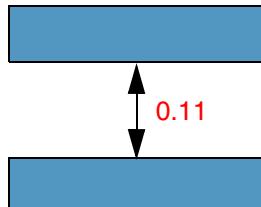
 Metal1



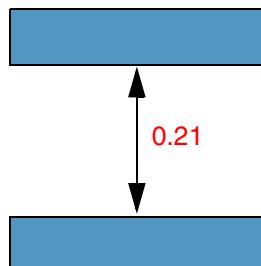
a) PASS. The spacing 0.1 falls in the allowed spacing range (when  $n=0$ ,  $S_{legal} = 0.1$  ( $0.1+0\times0.05$ )).



b) PASS. The spacing 0.15 falls in the allowed spacing range (when  $n=1$ ,  $S_{legal} = 0.15$  ( $0.1+1\times0.05$ )).



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 Constraint Reference

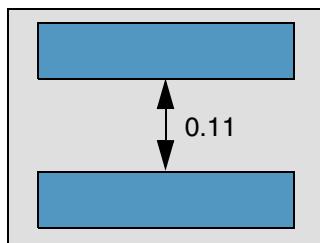
### Spacing Constraints (One Layer)

#### ***Example 5: allowedSpacingRanges with overLayer***

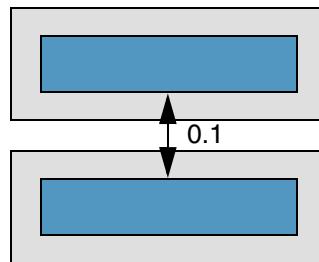
The spacing between two Metal1 shapes must be greater than 0.1 if both shapes overlap a single Metal2 shape.

```
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.

## cornerFillSpacing

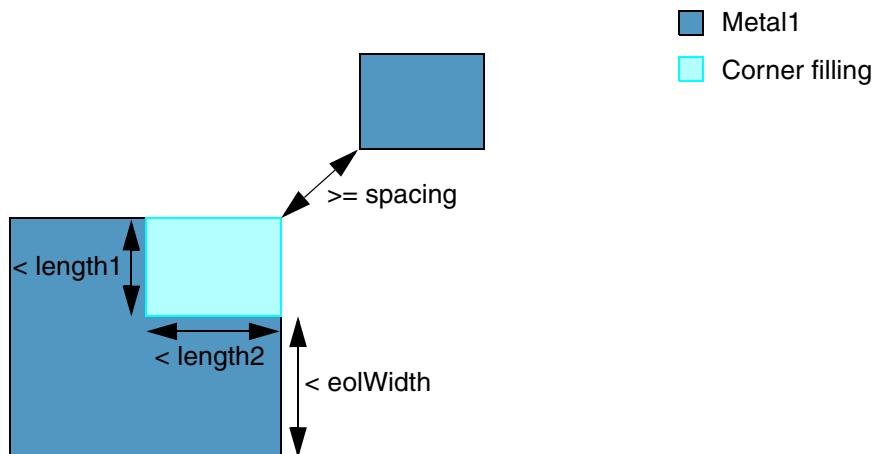
```
spacings(
  ( cornerFillSpacing tx_layer
    'width f_eolwidth
    'cornerLengths (f_length1 f_length2)
    f_spacing
  )
) ;spacings
```

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*.



## 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

'width *f\_eolwidth*

The edge of the concave corner with length *length2* must have an adjoining end-of-line edge with width less than *eolWidth*.

'cornerLengths (*f\_length1 f\_length2*)

The constraint applies if the length of one edge of the concave corner is less than *length1* and the length of the other edge is less than *length2*. The edge with length *length2* must have an adjoining end-of-line edge with width less than *eolWidth*.

## 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.
- The edge adjoining the concave edge with length less than 0.08 is an end-of-line edge with width less than 0.1.

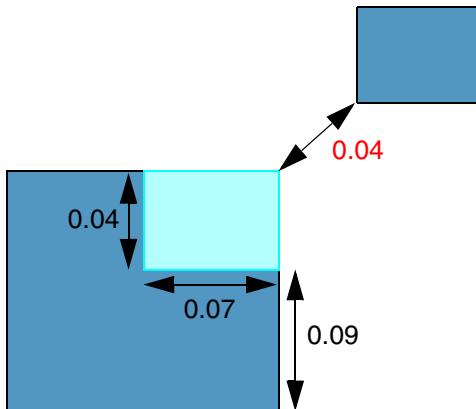
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

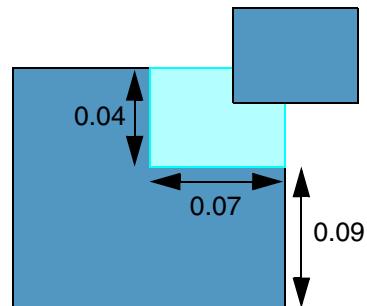
---

```
spacings(
  ( cornerFillSpacing Metal1
    'width 0.1
    'cornerLengths (0.06 0.08)
    0.05
  )
) ;spacings
```

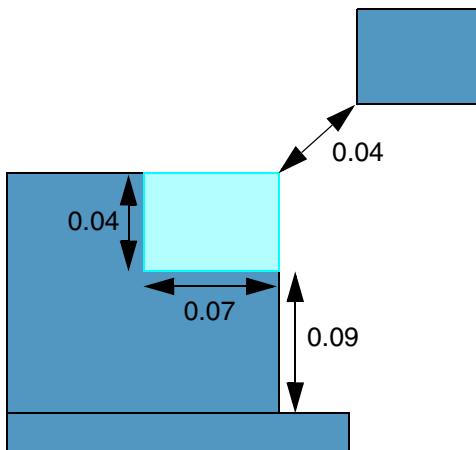
 Metal1  
 Corner filling



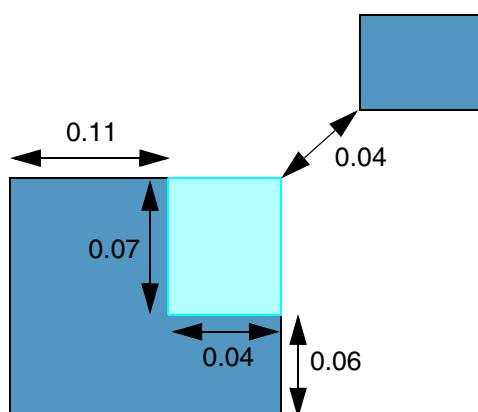
a) FAIL. Both required conditions are met, but the spacing is only 0.04 (<0.05).



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).

## endOfLineKeepout

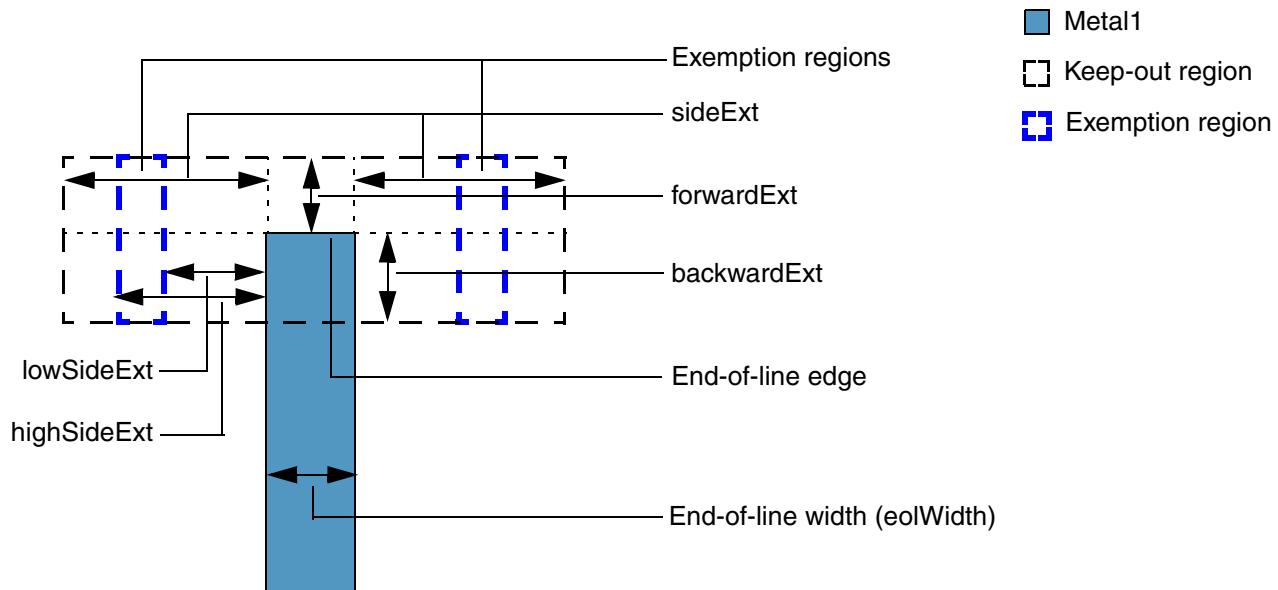
```

spacings(
    ( endOfLineKeepout tx_layer 'width f_width
        ['cornerOnly
            [ { 'exceptFromFrontEdge
                | 'exceptFromBackEdge ['forwardGap f_forwardGap]
            }
            [ [ 'mask1 | 'mask2 | 'mask3] ['twoSides] ['diffMask]
        ]
    ]
    ['exceptSameMetal] ['exceptSideWithin f_withinRange]
    ['orGroupID x_orGroupID ['otherEndEol]]
    (f_backwardExt f_sideExt f_forwardExt)
)
) ;spacings

```

Specifies the keep-out 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 keep-out region, 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 keep-out region, while in other processes, neither a corner nor an edge of another shape can intersect the keep-out region. However, in some processes, an intersecting corner or edge of a contiguous same-metal shape is not considered a violation.



## Virtuoso Technology Data Constraint 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>f_backwardExt</code>	The backward extension value of the keep-out region.
<code>f_sideExt</code>	The side extension value of the keep-out region.
<code>f_forwardExt</code>	The forward extension value of the keep-out region.

#### Parameters

<code>'width f_width</code>	The constraint applies to shapes with end-of-line width less than this value.
<code>'cornerOnly</code>	Only a corner inside the keep-out region is a violation. Otherwise, an edge or a corner of a neighboring shape intersecting the keep-out region is a violation. Type: Boolean
<code>'exceptSameMetal</code>	The corner or edge of a contiguous same-metal shape intersecting the keep-out region is not a violation. Type: Boolean
<code>'exceptFromFrontEdge</code>	(Advanced Nodes Only) Neighboring shapes that intersect the front (top) edge of the keep-out region do not cause a violation. Type: Boolean
<code>'exceptFromBackEdge</code>	(Advanced Nodes Only) Neighboring shapes that intersect the back (bottom) edge of the keep-out region do not cause a violation. Type: Boolean
<code>'forwardGap f_forwardGap</code>	(ICADV12.3 Only) If the spacing between the shape that is face-to-face with the end-of-line shape (inside the search window defined by <code>backwardExt</code> , <code>sideExt</code> , and <code>forwardGap</code> ) is less than the <code>forwardGap</code> value, then it must be equal to the <code>forwardExt</code> value specified for the constraint.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'mask1 | 'mask2 | 'mask3

(ICADV12.3 Only) The constraint applies only if the end-of-line shape is on the specified mask.

Type: Boolean

'twoSides

(ICADV12.3 Only) The constraint applies only if the end-of-line shape has two neighboring shapes, one on each side, in the keep-out region.

Type: Boolean

'diffMask

(ICADV12.3 Only) The constraint applies only if the end-of-line shape is on one mask and the two neighboring shapes are on another mask.

Type: Boolean

'exceptSideWithin *f\_withinRange*

(Advanced Nodes Only) 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.

'orGroupID *x\_orGroupID*

(Advanced Nodes Only) 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

(ICADV12.3 Only) If the edge found inside the keep-out 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

## Examples

- [Example 1: endOfLineKeepout](#)
- [Example 2: endOfLineKeepout with cornerOnly and exceptFromFrontEdge](#)
- [Example 3: endOfLineKeepout with exceptSideWithin](#)
- [Example 4: endOfLineKeepout with forwardGap](#)
- [Example 5: endOfLineKeepout with orGroupID \(OR\)](#)
- [Example 6: endOfLineKeepout with orGroupID \(OR and AND\)](#)
- [Example 7: endOfLineKeepout with mask, twoSides, and diffMask](#)
- [Example 8: endOfLineKeepout with otherEndEol](#)

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

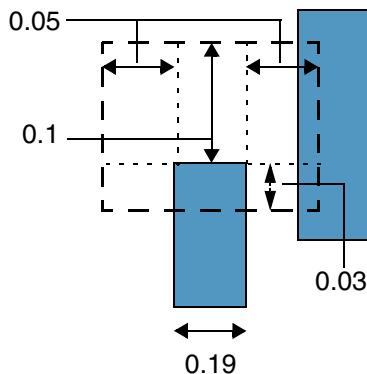
---

#### **Example 1: *endOfLineKeepout***

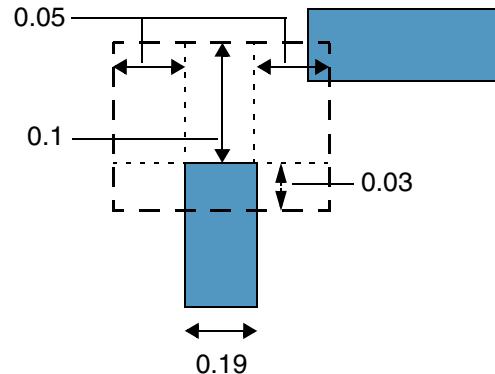
The keep-out region for a wire with end-of-line width less than 0.2 is defined by backward, side, and forward extension values of 0.03, 0.05, and 0.1, respectively. The constraint is violated if the corner or edge of a neighboring shape intersects the keep-out region.

```
spacings(
  ( endOfLineKeepout "Metall1" 'width 0.2
    (0.03 0.05 0.1)
  )
) ;spacings
```

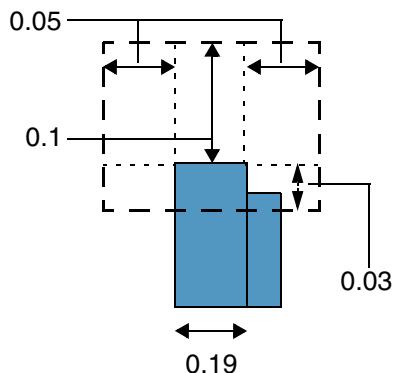
Metal1
Keep-out region



a) FAIL. The edge of a neighboring shape intersects the keep-out region. If 'cornerOnly' was defined, this would not be a violation.



b) FAIL. The corner of a neighboring shape intersects the keep-out region.



c) FAIL. Any intersecting neighboring shape, including a same-metal shape, is a violation.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

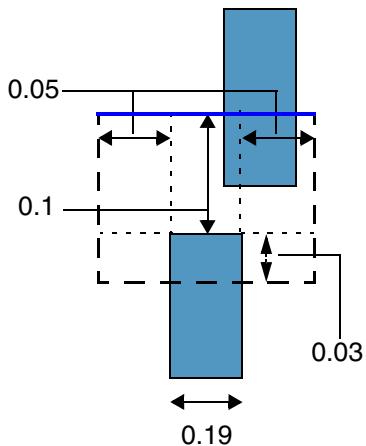
#### **Example 2: *endOfLineKeepout* with *cornerOnly* and *exceptFromFrontEdge***

The keep-out 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 the corner of a neighboring shape, which is not coming from the front edge of the keep-out region, intersects the keep-out region.

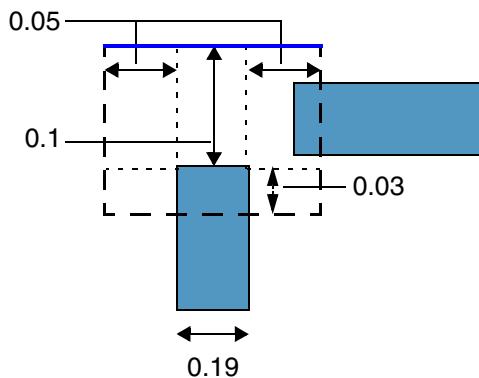
spacings (

```
( endOfLineKeepout "Metall1" 'width 0.2
    'cornerOnly
    'exceptFromFrontEdge
    (0.03 0.05 0.1)
)
) ;spacings
```

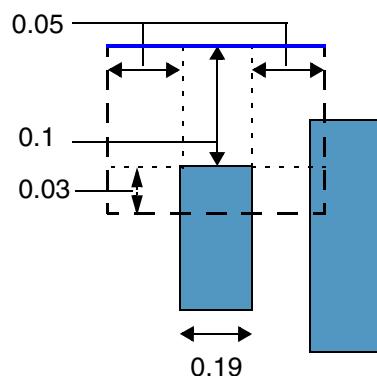
 Metal1  
 Keep-out region



a) PASS. The neighboring shape is coming from the front edge (the blue, solid line) of the keep-out region.



b) FAIL. The corner of a neighboring shape, which is not coming from the front edge, intersects the keep-out region.



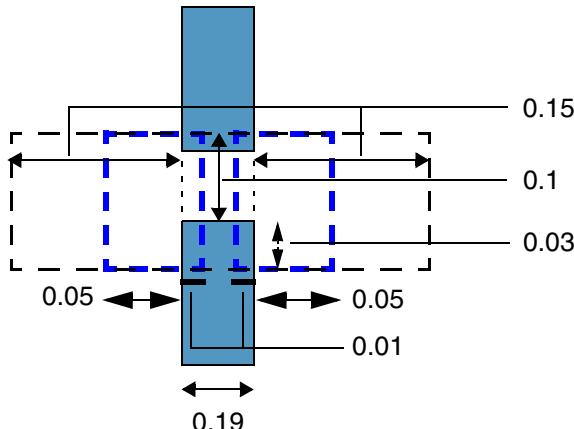
c) FAIL. The corner of a neighboring shape, which is not coming from the front edge, intersects the keep-out region.

**Example 3: *endOfLineKeepout* with *exceptSideWithin***

The keep-out 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 keep-out region, intersects the keep-out 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
- Keep-out region
- Exemption region



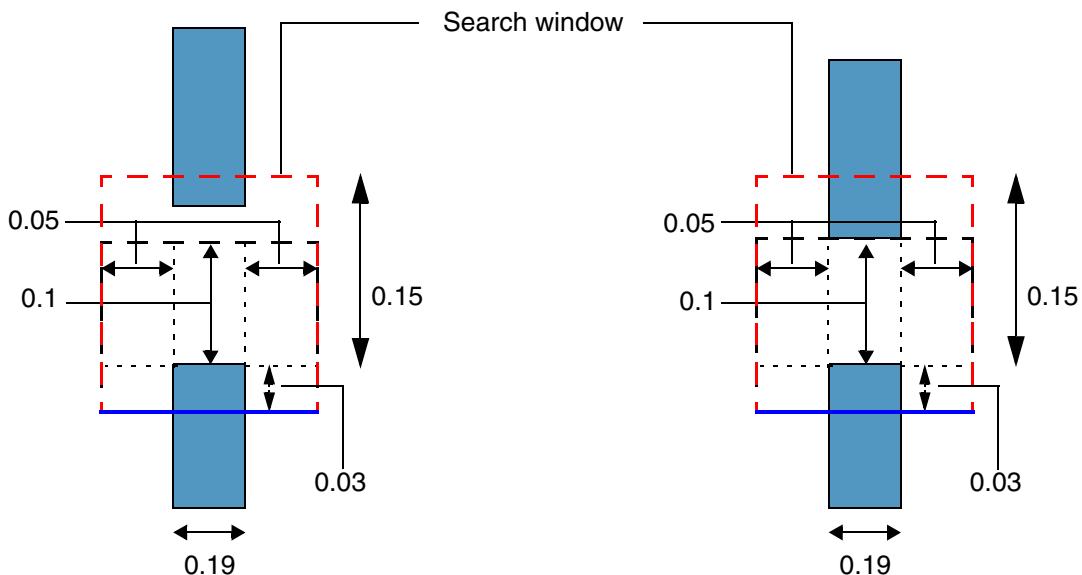
PASS. When '*exceptSideWithin*' is negative, a neighboring shape on the same track is exempted.

**Example 4: *endOfLineKeepout* with *forwardGap***

The keep-out 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 exactly 0.1 (the forward extension value).

```
spacings(
  ( endOfLineKeepout "Metall1" 'width 0.2
    'cornerOnly
    'exceptFromBackEdge 'forwardGap 0.15
    (0.03 0.05 0.1)
  )
) ;spacings
```

█ Metal1  
□ Keep-out 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 keep-out region.

b) PASS. The spacing between the two face-to-face shapes is equal to 0.1.

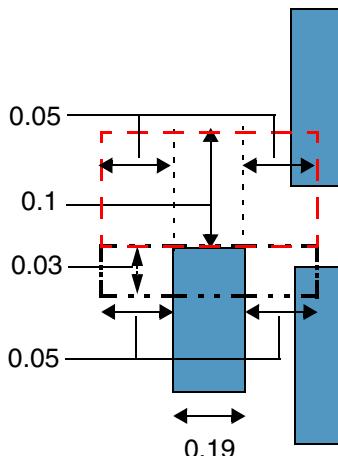
**Example 5: *endOfLineKeepout* with *orGroupID* (OR)**

Two keep-out 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:

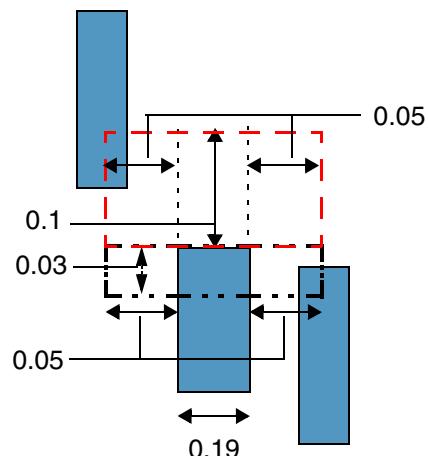
- The corner of a neighboring shape, which is not coming from the front edge of keep-out region 1, intersects keep-out region 1.
- The corner of a neighboring shape, which is not coming from the back edge of keep-out region 2, intersects keep-out region 2.
- Both neighboring shapes are on the same side of the end-of-line wire.

```
spacings(
    ( endOfLineKeepout "Metall1" 'width 0.2
        'cornerOnly
        'exceptFromFrontEdge
        'orGroupID 1
        (0.03 0.05 0.0)
    )
    ( endOfLineKeepout "Metall1" 'width 0.2
        'cornerOnly
        'exceptFromBackEdge
        'orGroupID 1
        (0.0 0.05 0.1)
    )
) ;spacings
```

<span style="background-color: #0070C0; border: 1px solid black; padding: 2px;"></span>	Metal1
<span style="border: 1px dashed black; padding: 2px;"></span>	Keep-out region 1
<span style="border: 1px dashed red; padding: 2px;"></span>	Keep-out 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 Constraint Reference

### Spacing Constraints (One Layer)

---

#### ***Example 6: 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:

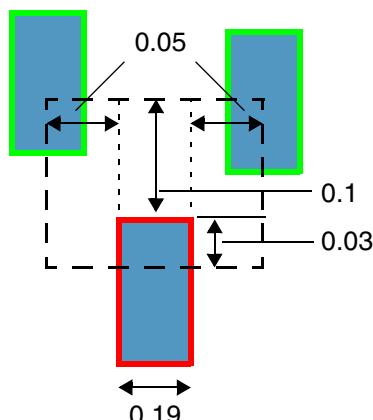
```
spacings(
  ( endOfLineKeepout "Metall1" 'width 0.06
    'cornerOnly
    'exceptFromFrontEdge
    'orGroupID 1
    (0.00 0.08 0.12)
  )
  ( endOfLineKeepout "Metall1" 'width 0.06
    'cornerOnly
    'exceptFromBackEdge
    'orGroupID 1
    (0.04 0.08 0.00)
  )
  ( endOfLineKeepout "Metall1" 'width 0.06
    'cornerOnly
    'orGroupID 1
    (0.04 0.06 0.12)
  )
  ( endOfLineKeepout "Metall1" 'width 0.09
    'cornerOnly
    'exceptFromFrontEdge
    'orGroupID 2
    (0.04 0.08 0.12)
  )
  ( endOfLineKeepout "Metall1" 'width 0.09
    'cornerOnly
    'exceptFromBackEdge
    'orGroupID 2
    (0.04 0.00 0.12)
  )
) ;spacings
```

**Example 7: *endOfLineKeepout* with *mask*, *twoSides*, and *diffMask***

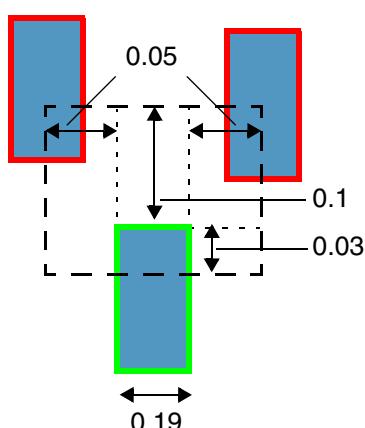
The keep-out region for a wire with end-of-line width less than 0.2 is defined by backward, side, and forward extension values of 0.03, 0.05, and 0.1, respectively. The constraint applies only to a mask1 wire with two neighboring wires on a different mask, one on each side. The neighboring wires must also satisfy '*cornerOnly*' and '*exceptFromBackEdge*'.

```
spacings(
  ( endOfLineKeepout "Metal2" 'width 0.2
    'cornerOnly
    'exceptFromBackEdge
    'mask1 'twoSides 'diffMask
    (0.03 0.05 0.1)
  )
) ;spacings
```

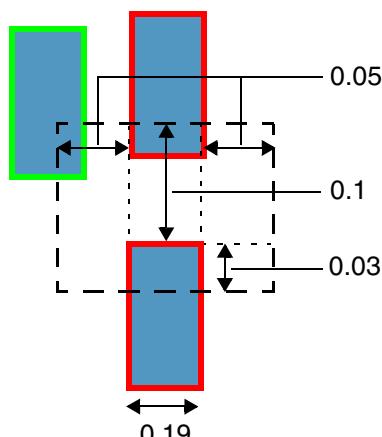
<span style="background-color: #4682B4; border: 1px solid black; padding: 2px;"></span> Metal1 <span style="background-color: #FF0000; border: 1px solid black; padding: 2px;"></span> mask1 <span style="background-color: #008000; border: 1px solid black; padding: 2px;"></span> mask2 <span style="border: 1px dashed black; padding: 2px;"></span> Keep-out 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 keep-out 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 Constraint Reference

### Spacing Constraints (One Layer)

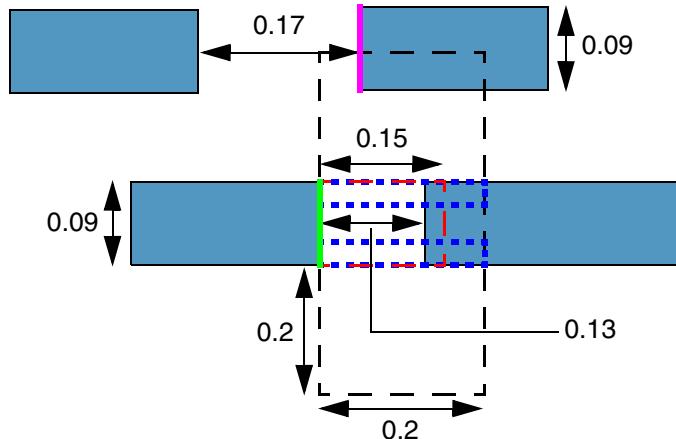
---

#### **Example 8: *endOfLineKeepout* with *otherEndEol***

Two *endOfLineKeepout* constraints in an OR group define two keep-out 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 keep-out region.

```
spacings(
  ( endOfLineKeepout "Metall1" 'width 0.1
    'cornerOnly
      'exceptFromBackEdge
      'exceptSideWithin (-0.01 0.0)
      'orGroupID 1 'otherEndEol
      (0.0 0.2 0.2)
  )
  ( endOfLineKeepout "Metall1" 'width 0.1
    'orGroupID 1
    (0.0 0.0 0.15)
  )
) ;spacings
```

- █ Metal1
- Keep-out region 1
- Keep-out region 2
- Exemption region



PASS. The pink edge is an end-of-line edge found inside the keep-out 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 keep-out region 1 (in black), and, therefore, fails the first *endOfLineKeepout* constraint; the bottom wire on the right intersects keep-out region 2 (in red), and, therefore, fails the second *endOfLineKeepout* constraint. Because the "other" end-of-line edge (in pink) inside the keep-out 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.

## **forbiddenEdgePitchRange (Advanced Nodes Only)**

```
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
    ]
    ['secondForbiddenRange g_secondForbiddenRange]
    ['sameMask]
    ['exactWireWidth]
    ['otherWidth f_otherWidth]
    (g_range)
  )
) ;spacings
```

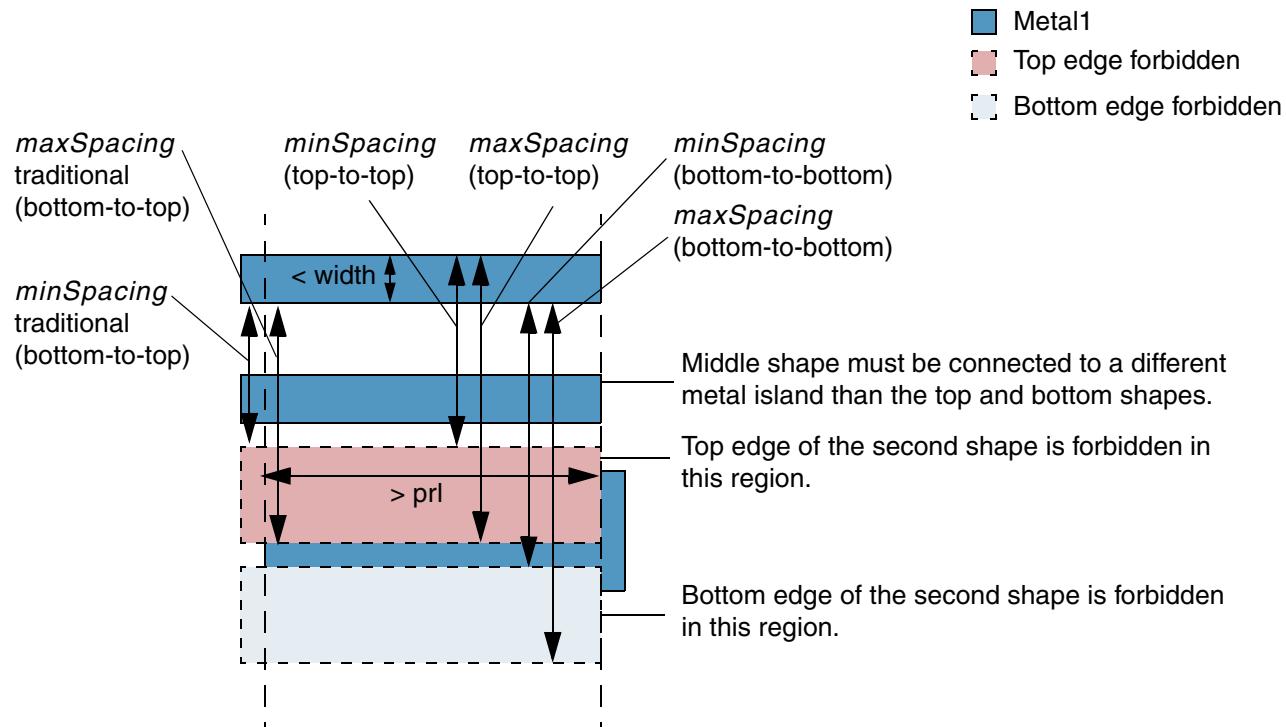
Restricts the spacing between the left-to-left and right-to-right edges (or between top-to-top and bottom-to-bottom edges) of two shapes on a layer if a shape from a different metal island lies between the two shapes. The constraint applies only if the width of the first shape is less than the specified 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 Constraint 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 Constraint Reference

### Spacing Constraints (One Layer)

---

#### Parameters

'width *f\_width*      The constraint applies only if the width of the first shape is less than this value.

'minWidth *f\_minWidth*      (ICADV12.3 Only) The constraint applies to a shape with width less than *width* and greater than *minWidth* if the shape is sandwiched by two shapes. The forbidden spacing applies between the middle shape and the outer shapes.

'outerWidth *f\_outerWidth*      (ICADV12.3 Only) The constraint applies only if the width of the two outer shapes is greater than this value.

'outerWithin *f\_outerWithin*      (ICADV12.3 Only) The constraint applies only if the distance between the two outer shapes is less than this value.

'outerSameMask      (ICADV12.3 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

'prl *f\_prl*      The constraint applies only if the parallel run length between the two edges and the in-between shape is greater than this value.

'parallelWithin *f\_parallelWithin*

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).

'exactSpacingNeighbor *f\_exactSpacing*

The constraint applies only if the first shape has a neighboring shape at a distance exactly 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.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'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.

'secondForbiddenRange *g\_secondForbiddenRange*

A second forbidden spacing range.

The forbidden spacing range is specified using the format:

[*f\_min2 f\_max2*]

where, *f\_min2* <= spacing <= *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

'exactWireWidth

The constraint applies only if the width of the first shape is exactly 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 Constraint Reference

### Spacing Constraints (One Layer)

---

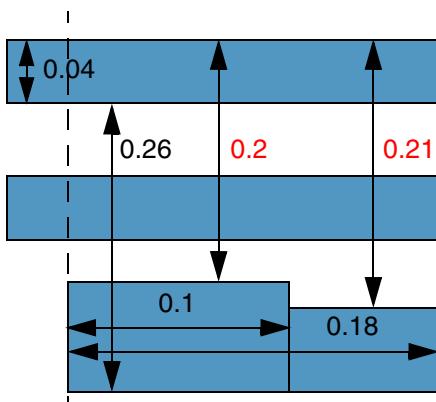
#### **Example 1: *forbiddenEdgePitchRange* with *width* and *prl***

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.

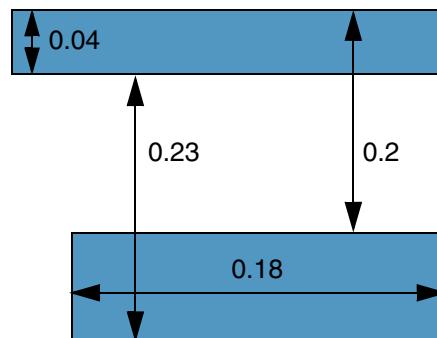
spacings (

```
( forbiddenEdgePitchRange "Metal1"
    'width 0.05 'prl 0.15
    "[0.2 0.25]"
)
) ; spacings
```

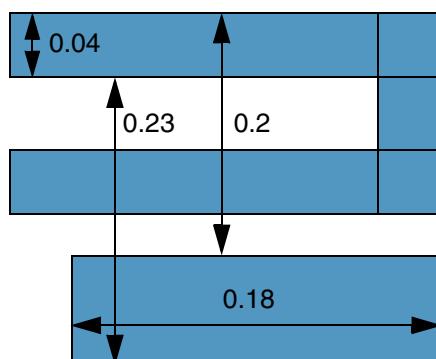
 Metal1



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).

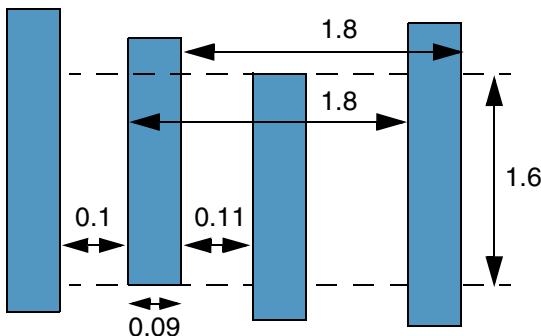
**Example 2: *forbiddenEdgePitchRange* with *width*, *prl*, and *neighborSpacingRanges***

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:

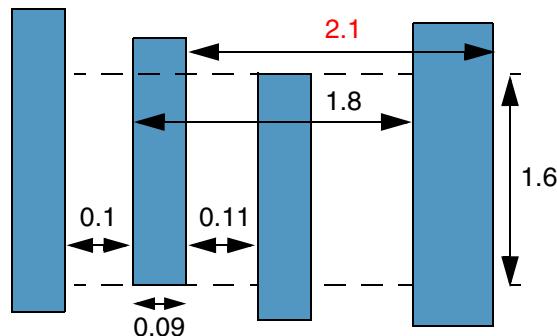
- The Metal1 wire under consideration has width less than 0.1.
- The parallel run length between the two Metal1 wires is greater than 1.5
- A wire from a different metal island with width equal to 0.1 or 0.15 or greater than or equal to 0.2 and less than or equal to 0.3 lies between the two Metal1 wires.

```
spacings(
  ( forbiddenEdgePitchRange "Metal1"
    'width 0.1 'prl 1.5
    'neighborSpacingRanges (0.1 0.15 "[0.2 0.3]"
    "[2.0 3.0]"
  )
) ;spacings
```

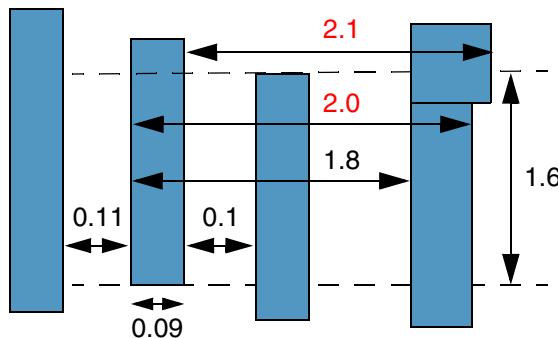
■ Metal1



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.



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 Constraint Reference

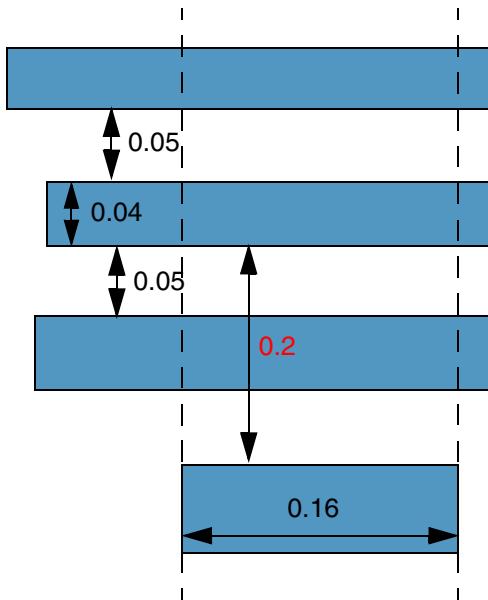
### Spacing Constraints (One Layer)

---

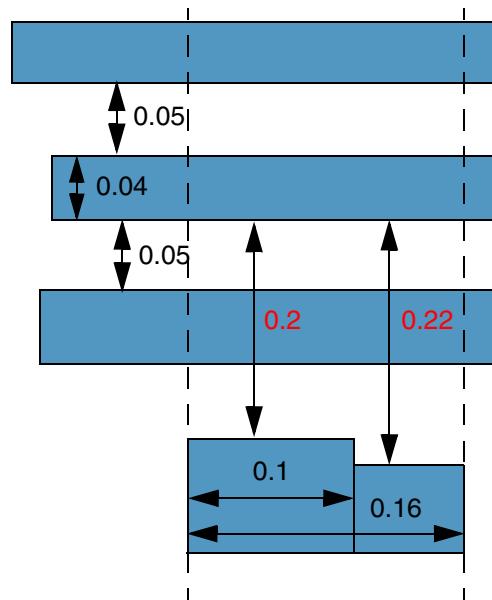
spacings(

```
( forbiddenEdgePitchRange "Metal1"
  'width 0.05 'prl 0.15
  'parallelWithin 0.06
  'secondForbiddenRange "[0.3 0.33]"
  "[0.2 0.25]"
)
) ; spacings
```

Metal1



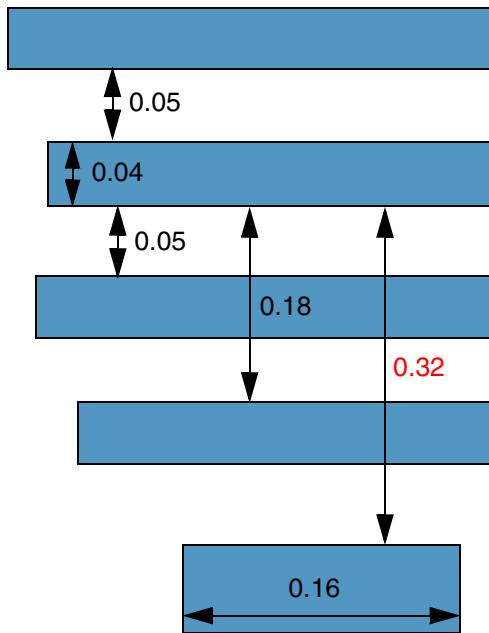
a) FAIL. The target wire (second from top) is 0.04 ( $<0.05$ ) wide and has two neighboring wires at a distance of 0.05 ( $<0.06$ ) from it. The parallel run length between the wires is 0.16 ( $>0.15$ ). However, the spacing between the target wire and the bottommost wire is 0.2, which lies in the forbidden range.



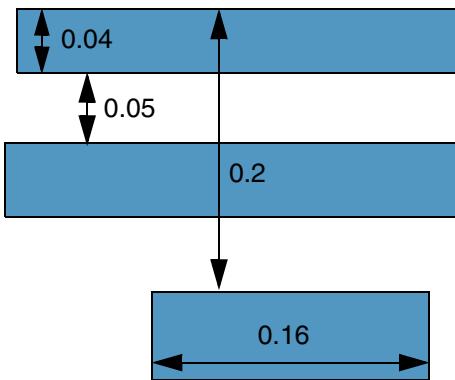
b) FAIL. The constraint applies. However, the spacing between the target wire (second from top) and the bottommost wire is 0.2 and 0.22, which lie in the forbidden range.

## Virtuoso Technology Data Constraint 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 Constraint Reference

### Spacing Constraints (One Layer)

---

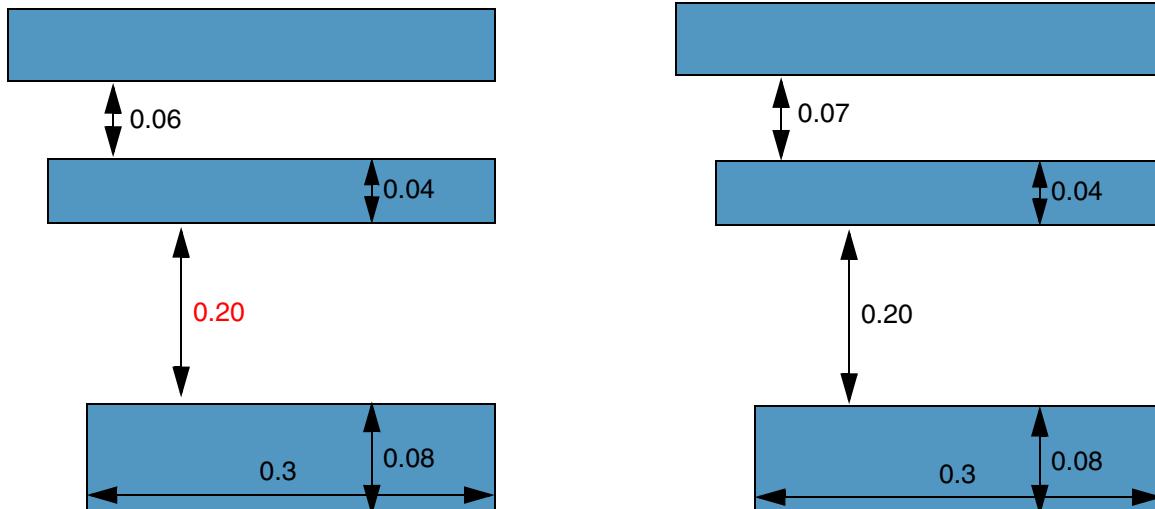
#### **Example 4: *forbiddenEdgePitchRange* with *width*, *prl*, *exactSpacingNeighbor*, and *spanLength***

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 exactly 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(
  ( forbiddenEdgePitchRange "Metall1"
    'width 0.05 'prl 0.04
    'exactSpacingNeighbor 0.06 'spanLength 0.09
    "[0.20 0.25]"
  )
) ;spacings
```

■ Metal1

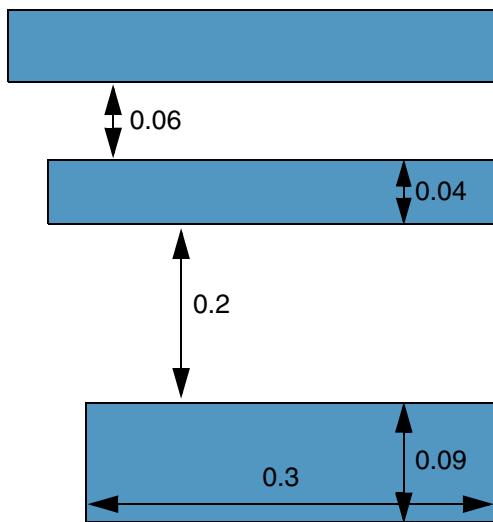


a) FAIL. The wire with width 0.04 (<0.05) has a neighbor that is exactly 0.06 away from it and the span length of the neighboring wire on the other side is 0.08 (<0.09). However, the spacing between the two wires is 0.20, which falls in the forbidden spacing range.

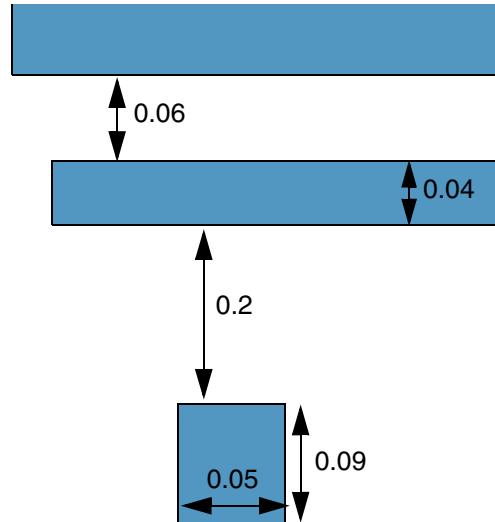
b) The constraint does not apply because the wire with width 0.04 (<0.05) does not have an exact spacing of 0.06 with the neighboring wire (the topmost wire).

## Virtuoso Technology Data Constraint 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).

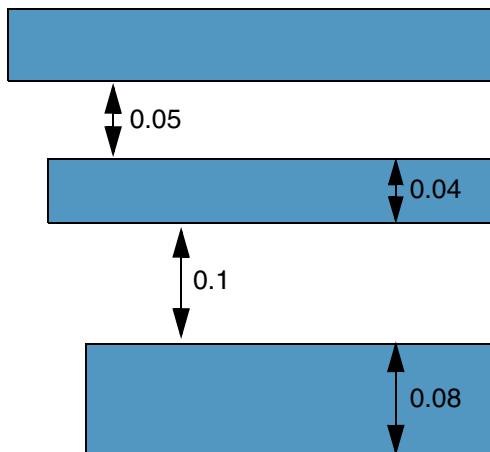
**Example 5: *forbiddenEdgePitchRange* with *width*, *exactSpacingNeighbor*, *exactWireWidth*, and *otherWidth***

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:

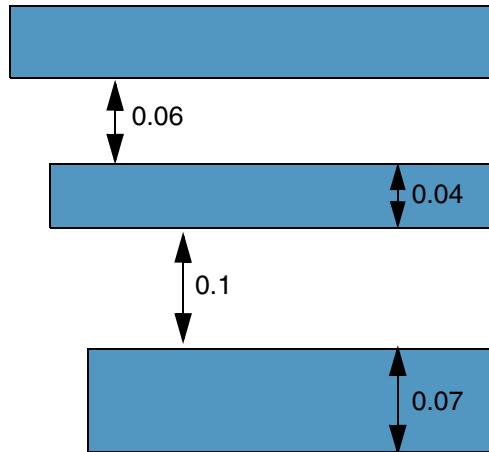
- The width of the first wire is exactly equal to 0.04 and has a neighboring wire at a distance exactly equal to 0.06 from it.
- The width of the second wire is greater than or equal to 0.08.

```
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).

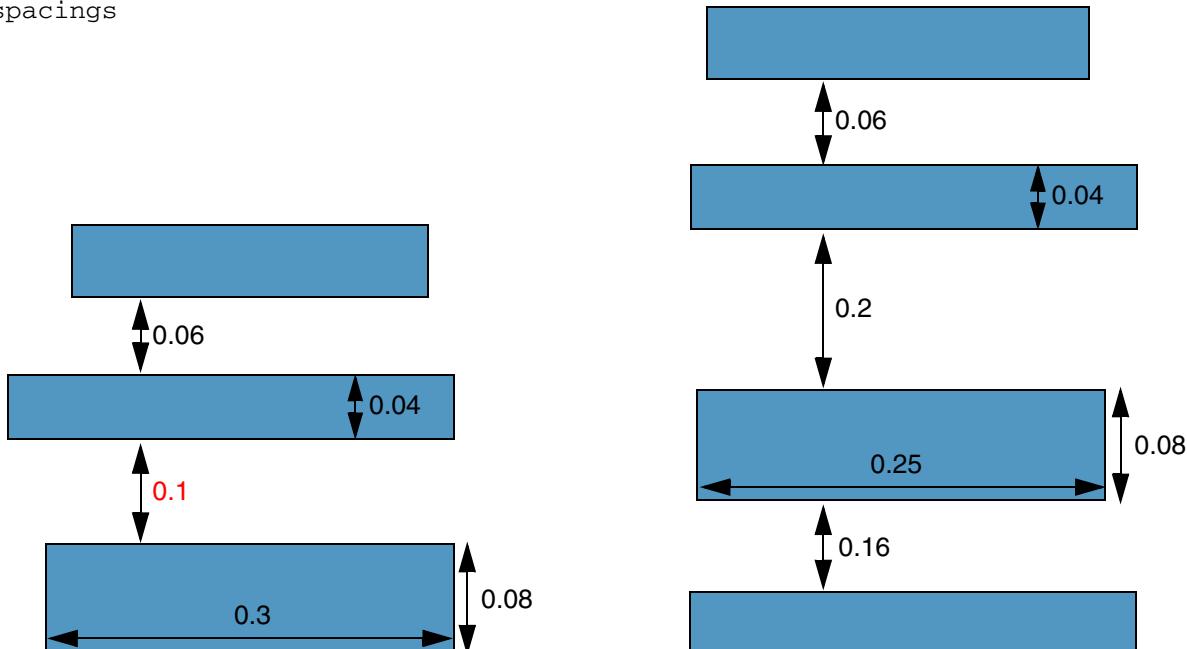
**Example 6: *forbiddenEdgePitchRange* with *width*, *prl*, *exactSpacingNeighbor*, *spanLength*, and *within***

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 exactly 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 ( $\geq 0.08$  and  $\leq 0.12$ ).

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.

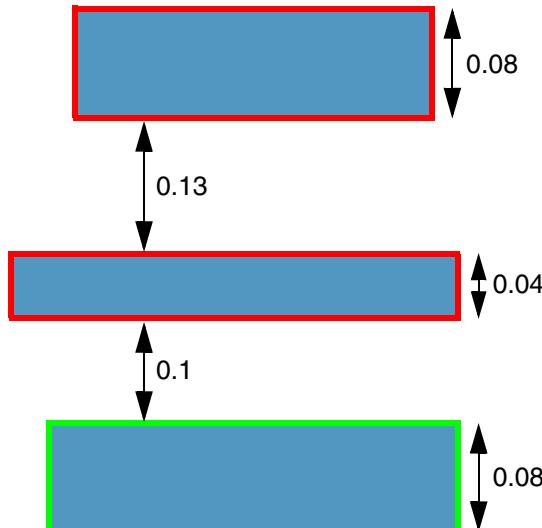
**Example 7: *forbiddenEdgePitchRange* with *width*, *minWidth*, *outerWidth*, *outerWithin*, *sameMask***

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
```

<span style="background-color: #4682B4; border: 1px solid black; width: 10px; height: 10px;"></span>	Metal1
<span style="background-color: #FF0000; border: 1px solid black; width: 10px; height: 10px;"></span>	Mask1
<span style="background-color: #008000; border: 1px solid black; width: 10px; height: 10px;"></span>	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 ( $\geq 0.08$  and  $\leq 0.12$ ). 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.

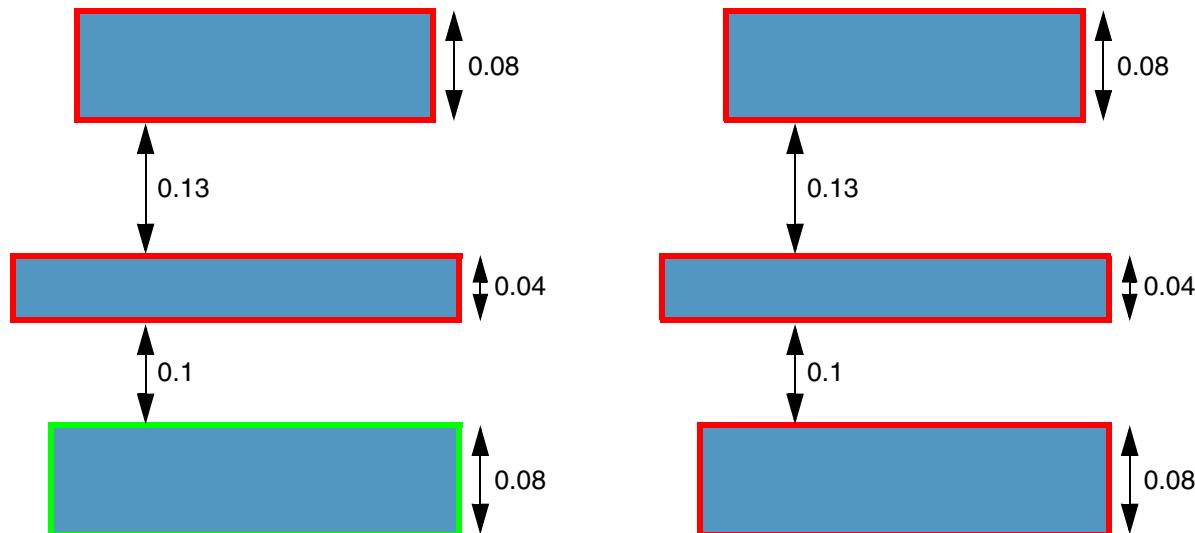
**Example 8: *forbiddenEdgePitchRange* with *width*, *minWidth*, *outerWidth*, *outerWithin*, *outerSameMask***

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 "Metall1"
    'width 0.05
    'minWidth 0
    'outerWidth 0.07
    'outerWithin 0.35
    'outerSameMask
    "[0.08 0.12]"
  )
) ;spacings
```

<span style="background-color: #0070C0; border: 1px solid black; width: 10px; height: 10px; display: inline-block;"></span>	Metal1
<span style="background-color: #FF0000; border: 1px solid black; width: 10px; height: 10px; display: inline-block;"></span>	Mask1
<span style="background-color: #00FF00; border: 1px solid black; width: 10px; height: 10px; display: inline-block;"></span>	Mask2



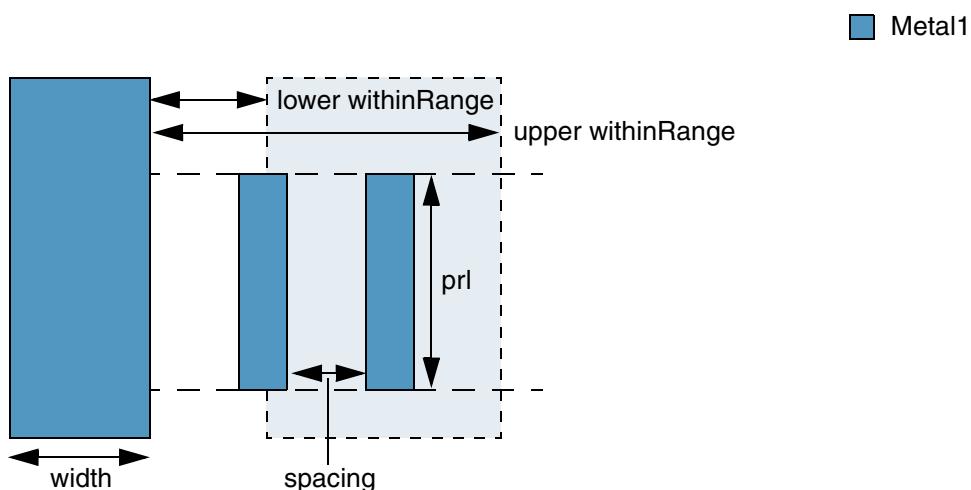
a) The constraint does not apply because the two outer wires are on different masks. A violation would occur if '*outerSameMask*' was not specified because the spacing between the middle and bottom wires lies in the forbidden spacing range ( $\geq 0.08$  and  $\leq 0.12$ ).

b) FAIL. The spacing between the middle and bottom wires lies in the forbidden spacing range ( $\geq 0.08$  and  $\leq 0.12$ ).

## forbiddenProximitySpacing (Advanced Nodes Only)

```
spacings(
  ( forbiddenProximitySpacing tx_layer
    'width f_width
    'prl f_prl
    'withinRange (g_withinRange)
    g_range
  )
) ; spacings
```

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.



## 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:  [ <i>f_min f_max</i> ]  where, <i>f_min</i> <= spacing <= <i>f_max</i>  Type: Floating-point values specifying a <u>range</u> of spacings that are not allowed.

## Parameters

'width <i>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.
'prl <i>f_prl</i>	The constraint applies only if the parallel run length common between the three neighboring shapes is greater than this value.
'withinRange ( <i>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 Constraint Reference

### Spacing Constraints (One Layer)

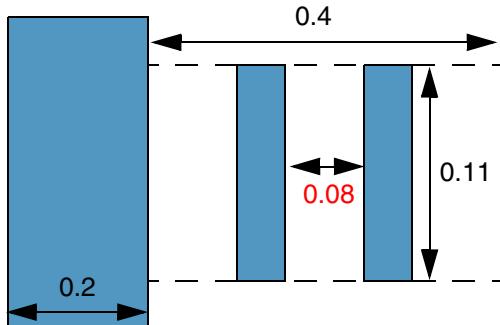
#### Example

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:

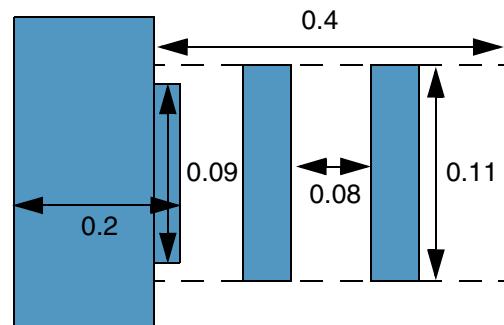
- The two wires are at a distance less than or equal to 0.4 from a wire with width greater than or equal to 0.2.
- The parallel run length common between the three neighboring wires is greater than 0.1.

```
spacings(  
  ( forbiddenProximitySpacing "Metall1"  
    '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.

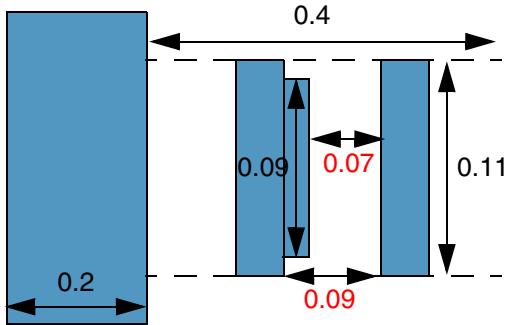


b) The constraint does not apply because the part of the wide wire with width 0.2 has parallel run length of only 0.09 ( $<0.1$ ) with the neighboring wires.

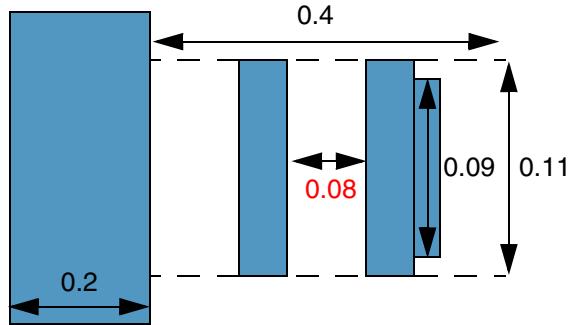
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

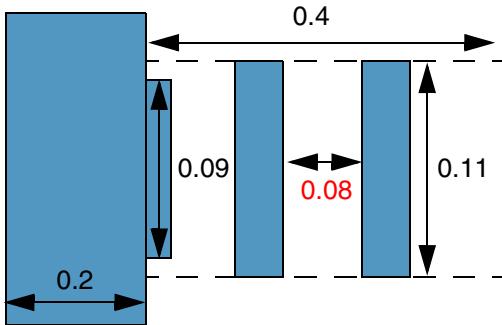
---



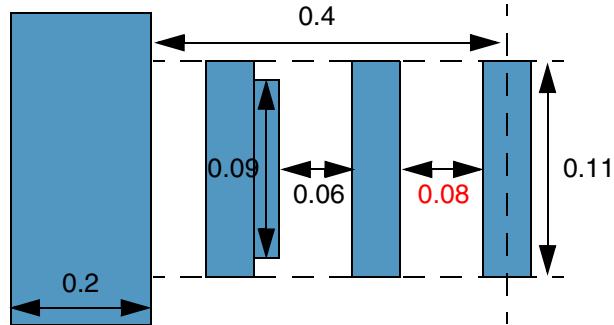
c) FAIL. The two wires are at a distance less than or equal to 0.4 from a wire with width 0.2 and the overall parallel run length common to the three wires is 0.11 ( $>0.1$ ). However, the spacings of 0.07 at the jog and 0.09 for the rest of the wire fall in the forbidden range.



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.

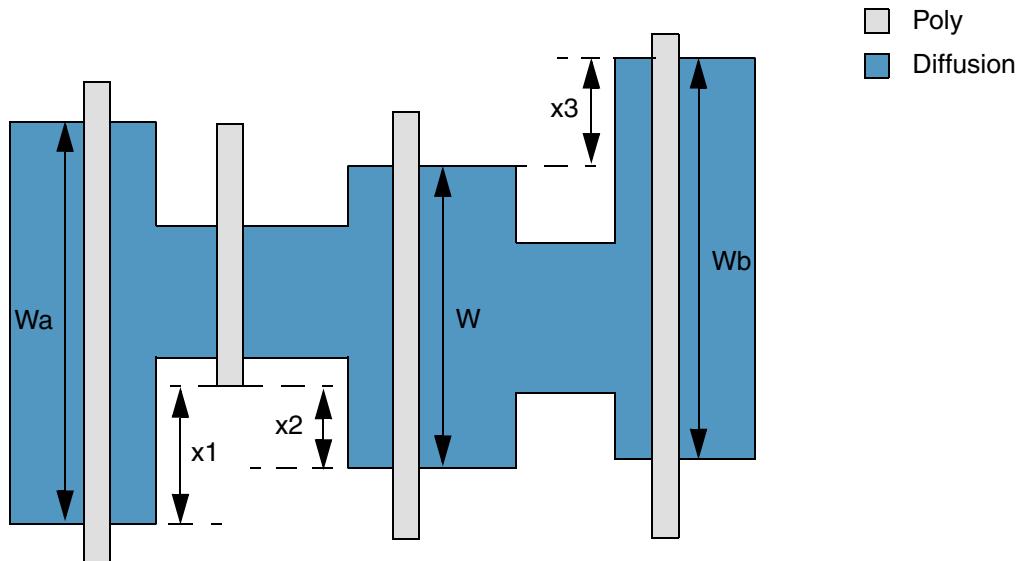
## gateSpacingRanges

```
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.



## 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 <u>ranges</u> between a gate and the neighboring poly shapes.

## Parameters

'ratio <i>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.
'length <i>f_length</i>	The constraint applies only if the channel length of the gate is less than or equal to this value.

## Example

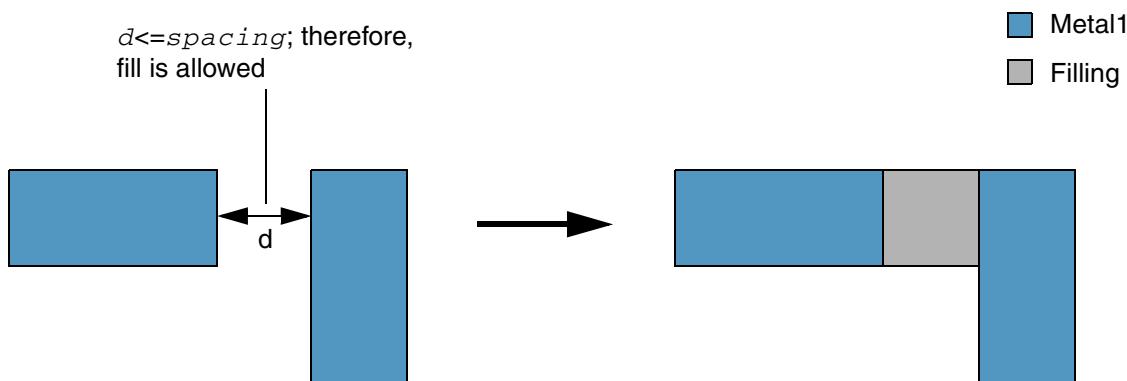
The spacing between a gate and a neighboring Poly shape must be greater than zero or less than 0.5 or greater than or equal to 0.1 if the width of the gate is less than or equal to 2 and the ratio of the distance between the top of the gate and a neighboring poly shape to the width of the gate is less than 0.25.

```
spacings(
  ( gateSpacingRanges "Poly"
    'ratio 0.25
    'length 2
    ("(0 0.5)" ">= 1")
  )
) ;spacings
```

## 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 Constraint Reference

### Spacing Constraints (One Layer)

---

#### Example

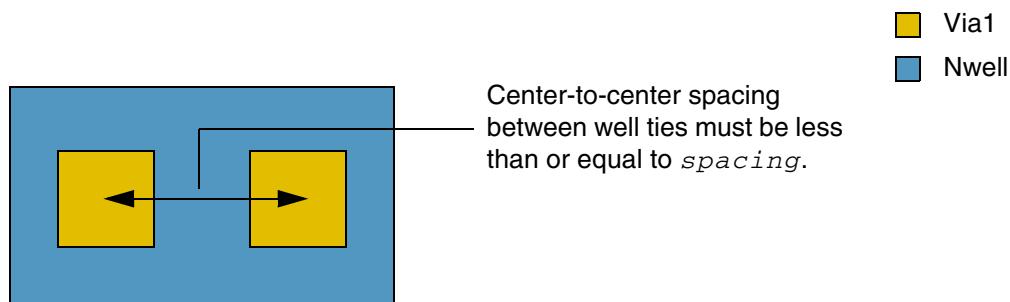
Fill is allowed if the spacing between shapes on Metal3 is less than or equal to 1.5.

```
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 Constraint 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.

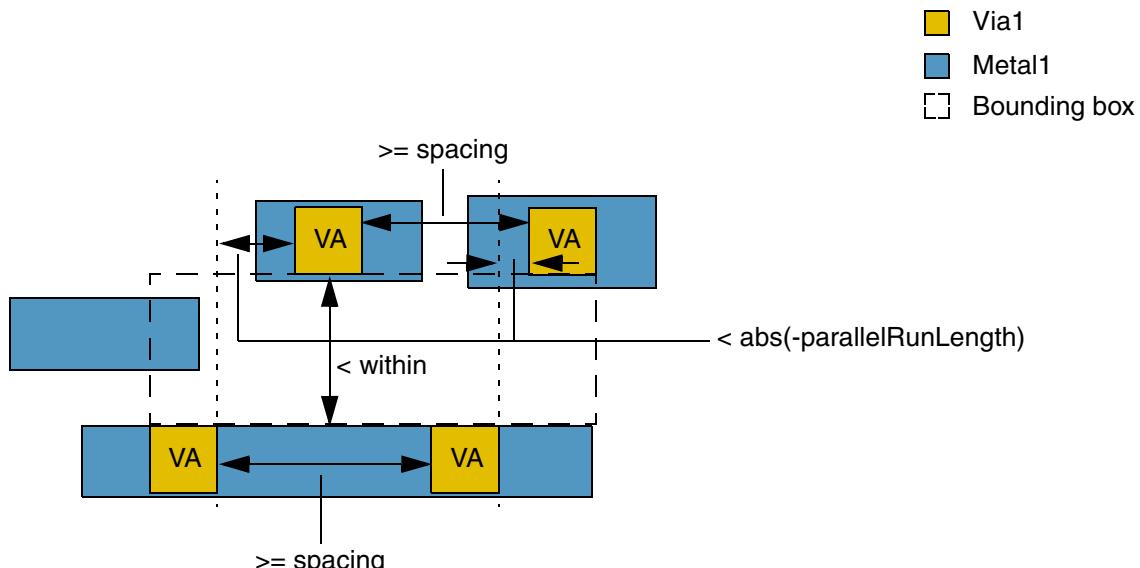
```
spacings(
  ( maxTapSpacing "Nwell"
    1.5
  )
) ; spacings
```

## minAdjacentFourViaSpacing (ICADV12.3 Only)

```
orderedSpacings(
    ( minAdjacentFourViaSpacing tx_metalLayer tx_cutLayer
        'cutClass {f_width | (f_width f_length) | t_name}
        'prl f_parallelRunLength 'within f_within
        ['horizontal | 'vertical]
        f_spacing
    )
)
; orderedSpacings
```

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.



## 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_spacing</i>	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*}

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

'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.

'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

## Example

The spacing between two adjacent via cuts must be greater than or equal to 0.07 if the following conditions are met:

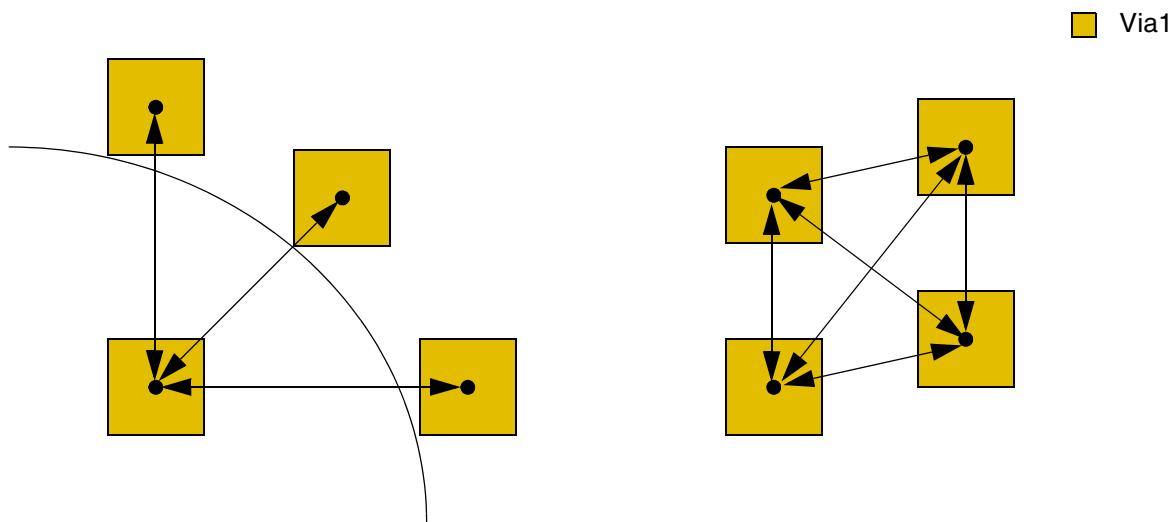
- Each via cut has a neighboring via cut at a distance less than 0.06 from it and has parallel run length greater than 0.05 with this neighboring via cut.
- A wire on Metal3 overlaps the bounding box formed by the via cut edges that face each other.

```
orderedSpacings(  
  ( minAdjacentFourViaSpacing "Metal3" "Via2"  
    'cutClass "VA"  
    'prl 0.05 'within 0.06  
    0.07  
  )  
) ;orderedSpacings
```

## 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

## 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) (Advanced Nodes Only)**

```
spacings(
  ( minClusterSpacing tx_layer
    'widthRange g_widthRange
    'numShapesRange g_numShapesRange
    'spacingRange g_spacingRange
    ['centerLineDistance]
    ['minSpaceOverride]
    ['horizontal | 'vertical]
    ['otherWidthRange g_otherWidthRange]
    ['groupHorizontal | 'groupVertical]
    ['prl f_prl]
    ['clusterToCluster]
    ['edgeToEdge]
    ['sameMask]
    f_spacing
  )
) ;spacings
```

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.

## 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

'widthRange <i>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.
'numShapesRange <i>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 that a cluster can contain.
'spacingRange <i>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.
'centerLineDistance	The spacing is measured from the centerline of the cluster to centerline of the neighboring non-cluster shapes.  <b>Note:</b> The intra-cluster spacing is always measured from the centerline, irrespective of whether this parameter is specified.  Type: Boolean

## Virtuoso Technology Data Constraint 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

'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.

'clusterToCluster

(ICADV12.3 Only) The constraint applies only between neighboring clusters. Otherwise, the constraint also applies between a cluster and neighboring non-cluster shapes.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

'edgeToEdge

(ICADV12.3 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 'centerLineDistance' parameter determines how spacing is measured between a cluster and neighboring non-cluster shapes.

Type: Boolean

' sameMask

(ICADV12.3 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

## 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
  - 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

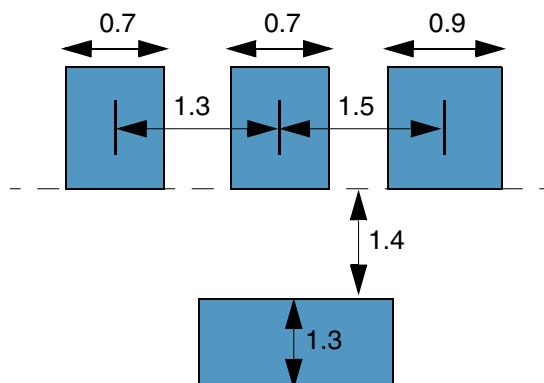
**Example 1: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, and *otherWidthRange***

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
```



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 Constraint Reference

### Spacing Constraints (One Layer)

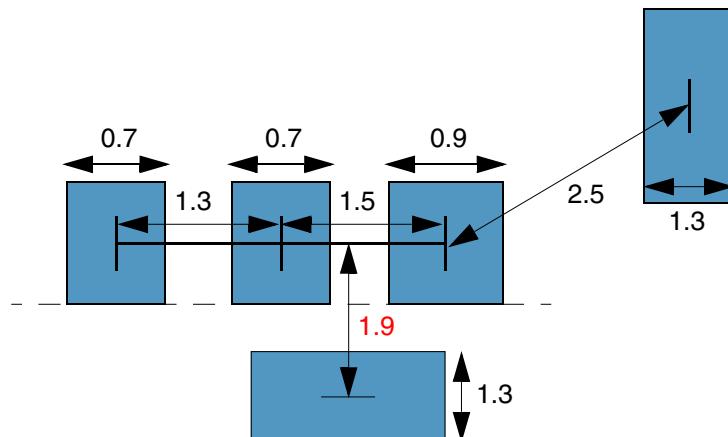
#### **Example 2: minClusterSpacing with widthRange, numShapesRange, spacingRange, centerLineDistance, and otherWidthRange**

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(           Metal1
  ( minClusterSpacing "Metal1"
    'widthRange [0.7 0.9]
    'numShapesRange [3 4]
    'spacingRange [1.3 1.5]
    'centerLineDistance
    'otherWidthRange [1.2 1.5]
    2.0
  )
) ;spacings
```



**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 Constraint Reference

### Spacing Constraints (One Layer)

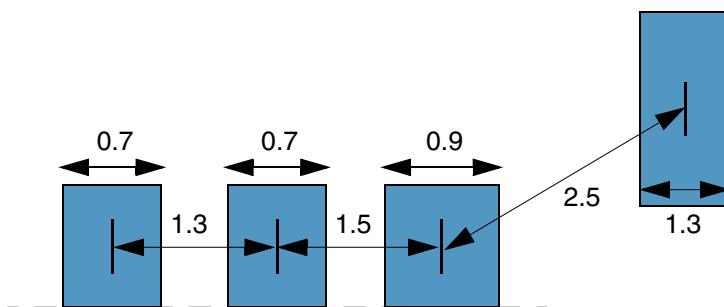
#### **Example 3: minClusterSpacing with widthRange, numShapesRange, spacingRange, centerLineDistance, otherWidthRange, and groupVertical**

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(                                Metal1
  ( 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
```



The constraint does not apply because the cluster direction is horizontal.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

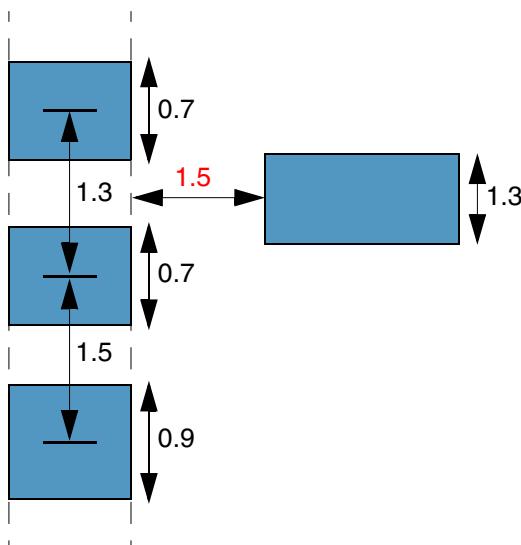
#### **Example 4: minClusterSpacing with widthRange, numShapesRange, spacingRange, horizontal, otherWidthRange, and groupVertical**

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].

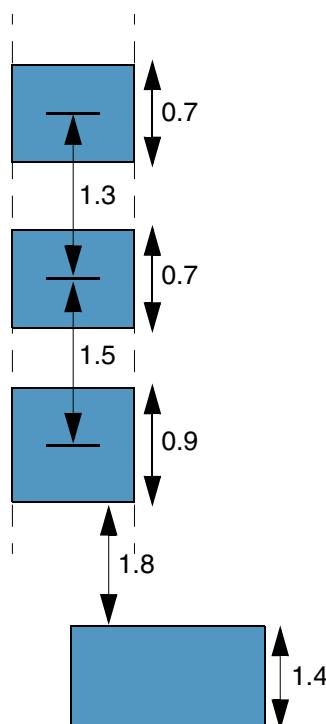
spacings(

```
( 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
```



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).

Metal1



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 Constraint 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:

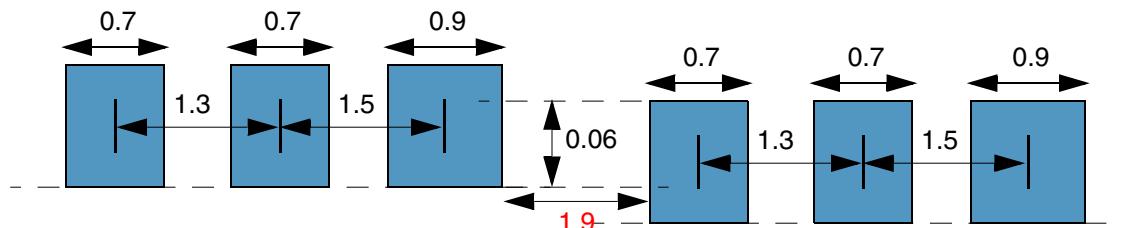
- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

If the neighboring shape belongs to a cluster, a minimum spacing requirement of 2.0 must be satisfied.

```

spacings(
  ( minSpacing "Metall1" 2.0 )
  ( minClusterSpacing "Metall1"
    '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 ( $>1.4$ ).

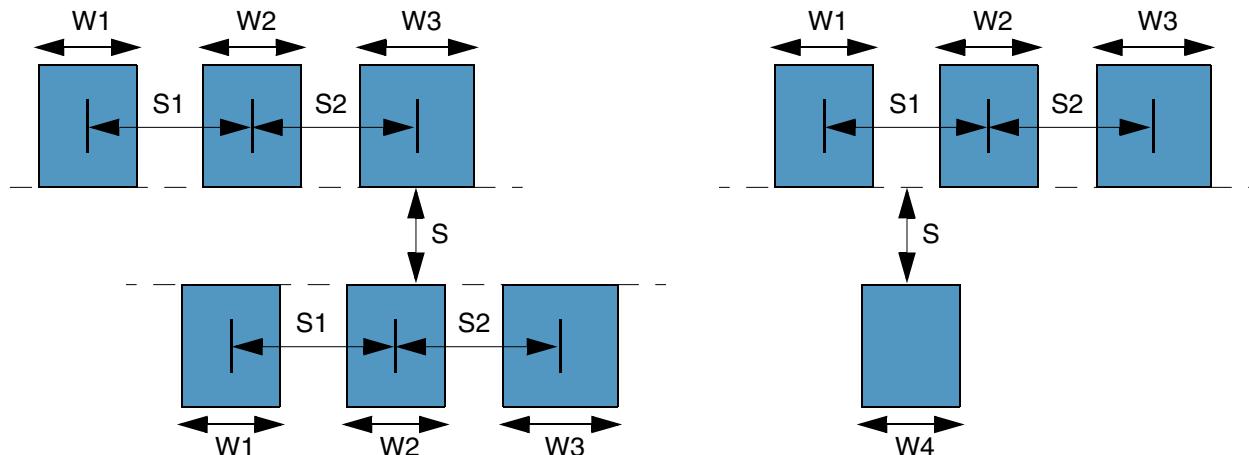
**Example 6: minClusterSpacing with widthRange, numShapesRange, spacingRange, and clusterToCluster**

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 "Metall1"
    'widthRange W
    'numShapesRange NR
    'spacingRange SR
    'clusterToCluster
    minS
  )
) ;spacings
```

Metal1



a) The constraint applies if  $w_1$ ,  $w_2$ , and  $w_3$  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.

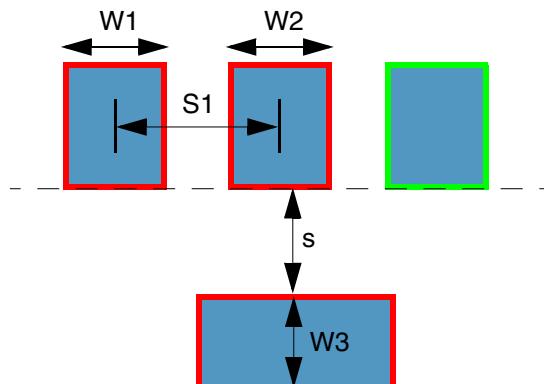
**Example 7: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *otherWidthRange*, and *sameMask***

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
```



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  $S < mins$ .

### **minConcaveCornerSpacing (ICADV12.3 Only)**

```
    spacingTables(
        ( minConcaveCornerSpacing tx_layer
            (( "width" nil nil )
                [ 'minLength f_minLength]
                [ 'exceptNotch | 'exceptNotchLength f_notchLength]
            )
            (g_table)
        )
    )
) ; spacingTables
```

Specifies a pair of spacing values that together 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,

1

- corresponding spacing values apply if the width of the neighboring wire is greater than or equal to this value.
  - $f_{horizSpacing}$  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.

## 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*

(ICADV12.3 Only) The constraint does not apply to a U-shaped notch if the notch length is less than this value.

## Examples

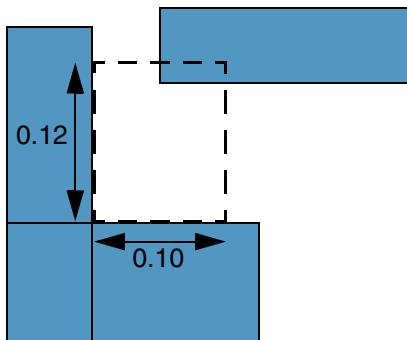
- [Example 1: minConcaveCornerSpacing with exceptNotch](#)
- [Example 2: minConcaveCornerSpacing with exceptNotchLength](#)

**Example 1: *minConcaveCornerSpacing* with *exceptNotch***

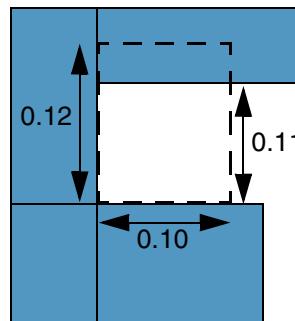
A neighboring wire must not be present inside a forbidden region that extends 0.1 horizontally and 0.12 vertically from a concave corner. The constraint does not apply to a U-shaped notch if the notch length is less than 0.1 (the smaller of the two spacing values).

```
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 Constraint Reference

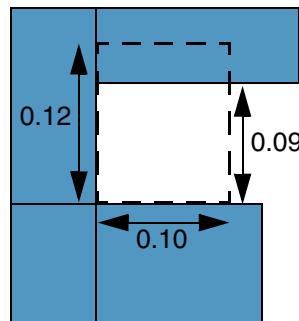
### Spacing Constraints (One Layer)

#### ***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)
        ['sameSideExtension f_extension
            | ['edgeForwardExtension f_edgeFwdExt
                'cornerExtensionRegion (f_cornerFwdExt f_cornerBkExt)
                'spanLength f_spanLength
                'minOppositeWidth f_minOppWidth
                'oppositeExtension (f_oppSideExt f_oppFwdExt1 f_oppFwdExt2)
            ]
        ]
        f_spacing
    )
) ;spacings

```

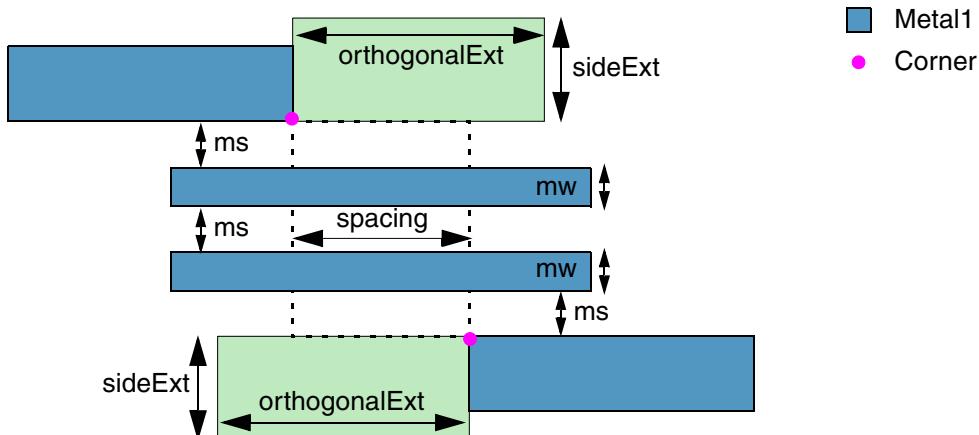
Specifies the minimum orthogonal spacing on a layer between two convex corners of two parallel shapes or between a convex corner and a neighboring shape.

The following are three scenarios in which the constraint applies:

### Scenario 1

The constraint applies between the convex corners of two parallel shapes facing each other if all of the following conditions are met:

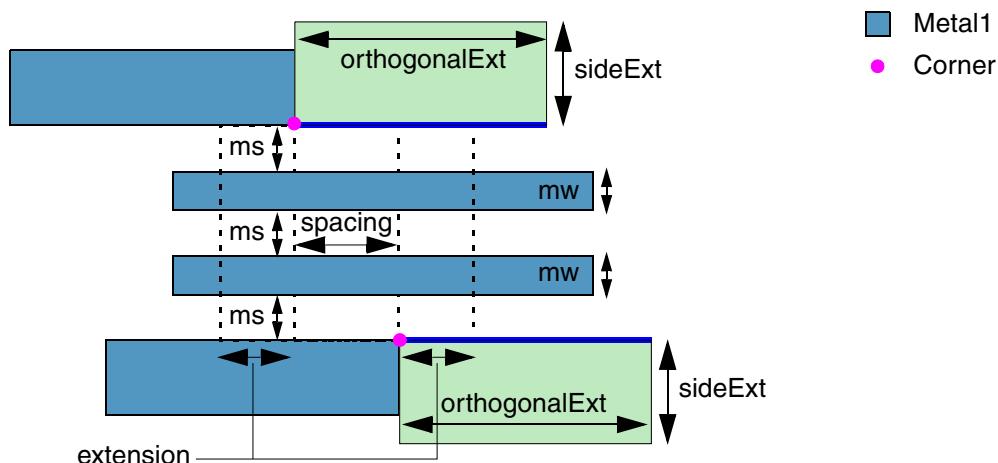
- The parallel shapes have between them two minimum-width (*mw*) shapes at minimum spacing (*ms*) from each other and from the parallel shapes, running across the region between the two convex corners.
- No neighboring shapes overlap the two exclusion regions defined by *sideExt* and *orthogonalExt* (green boxes).



## 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.

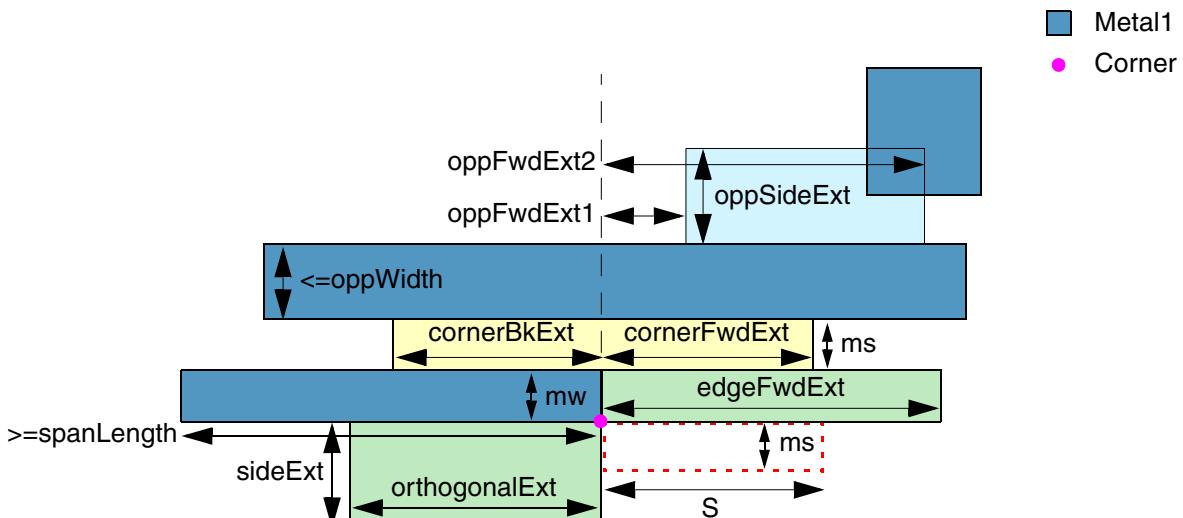


### Scenario 3

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.



## 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 ( <i>f_sideExt f_orthogonalExt</i> )	The constraint applies only if no neighboring wire overlaps the exclusion region defined by these values. <i>sideExt</i> is measured from the convex corner along the shorter side of the shape and <i>orthogonalExt</i> is measured perpendicular to the shorter side of the shape.
'sameSideExtension <i>f_extension</i>	The region between the two convex corners that do not face each other is extended on both sides by this value.
'edgeForwardExtension <i>f_edgeFwdExt</i>	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 <i>mw</i> .
'cornerExtensionRegion ( <i>f_cornerFwdExt f_cornerBkExt</i> )	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. <i>cornerFwdExt</i> is measured away from the shape with the convex corner and <i>cornerBkExt</i> is measured along the shape with the convex corner.
'spanLength <i>f_spanLength</i>	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.

## 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 1: minConvexCornerSpacing with sideExclusionRegion***

```
spacings()
  ( minConvexCornerSpacing "Metall1"
    'sideExclusionRegion (0.3 0.8)
    0.6
  ) ;spacings
```

### ***Example 2: minConvexCornerSpacing with sideExclusionRegion and sameSideExtension***

```
spacings()
  ( minConvexCornerSpacing "Metall1"
    'sideExclusionRegion (0.3 0.8)
    'sameSideExtension 0.2
    0.6
  ) ;spacings
```

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

**Example 3: *minConvexCornerSpacing* with *sideExclusionRegion*, *edgeForwardExtension*, *cornerExtensionRegion*, *spanLength*, *minOppositeWidth*, and *oppositeExtension***

```
spacings(
  ( minConvexCornerSpacing "Metall1"
    'sideExclusionRegion (0.3 0.8)
    'edgeForwardExtension 0.7
    'cornerExtensionRegion (0.6 0.6)
    'spanLength 1.3
    'minOppWidth 0.1
    'oppExtension (0.2 0.3 0.7)
    0.5
  ) ; spacings
```

## **minCornerSpacing (One layer) (Advanced Nodes Only)**

```
spacingTables(
  ( minCornerSpacing tx_layer
    (( "width" nil nil )
     {'convex
      ['endOfLineWidth f_width
       ['exceptJogLength f_jogLength
        ['useEdgeLength] ['treatLASJog]
        {'horizontalJog | 'verticalJog}
        'jogWidth (f_jogWidth f_wireWidth)
       ]
      ]
      ['sameMask]
      ['withinCornerOnly f_within {'horizontal | 'vertical}]
     | 'concave
      ['minLength f_minLength]
      ['exceptNotch | 'exceptNotchLength f_notchLength]
     }
     ['exceptSameNet | 'exceptSameMetal]
    )
   (g_table)
  )
) ;spacingTables
```

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.

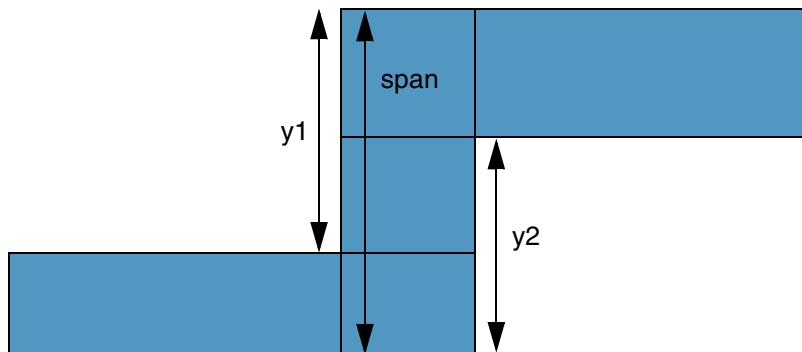
## Virtuoso Technology Data Constraint Reference

### 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 Constraint 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

(ICADV12.3 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 Constraint 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*.

'sameMask

(ICADV12.3 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

'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*

(ICADV12.3 Only) The constraint does not apply to a notch if the notch length is less than this value.

## Virtuoso Technology Data Constraint 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](#)

**Example 1: minCornerSpacing with convex, exceptEndOfLineWidth, concave, and exceptNotch**

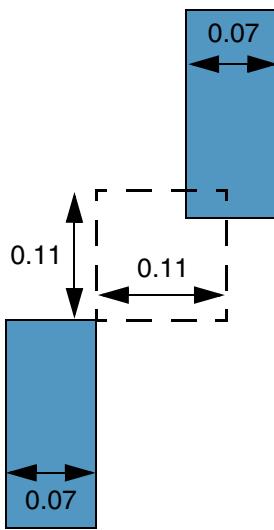
The minimum spacing requirement for a convex corner is 0.11, except when the two shapes have end-of-line widths less than 0.09 or the parallel run length between them is less than or equal to zero. The minimum spacing requirement for a concave corner is 0.15, except when a concave edge is less than 0.05 in length or the shape is a notch.

```

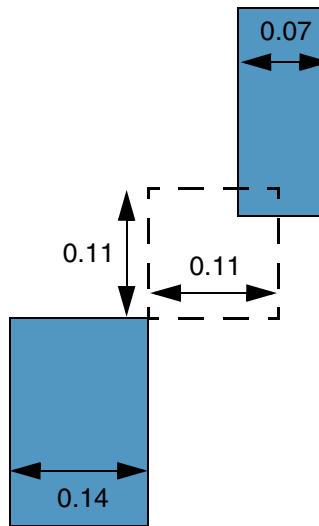
spacingTables(
  ( minCornerSpacing "Metal1"
    (( "width" nil nil )
     'convex
     'exceptEndOfLineWidth 0.09
    )
    (
      0.00  0.11
    )
  )
  ( minCornerSpacing "Metal1"
    (( "width" nil nil )
     'concave
     'exceptNotch
    )
    (
      0.00  0.15
    )
  )
) ; spacingTables

```

■ Metal1



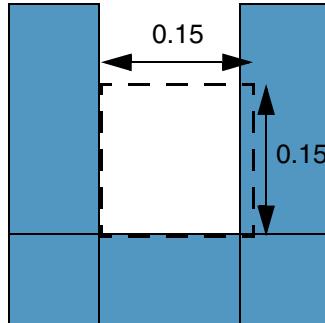
a) The constraint does not apply because the end-of-line width of both shapes is 0.07 (<0.09).



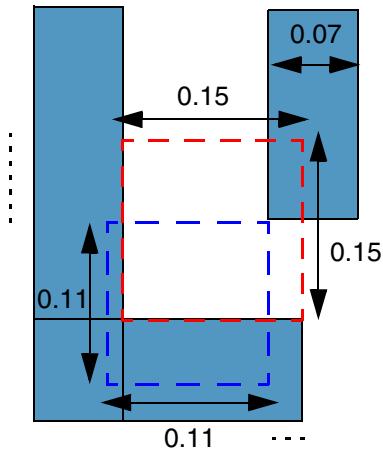
b) FAIL. The end-of-line width of one of the shapes is 0.14 (>0.09). Therefore, the spacing between the two shapes must be at least 0.11.

## Virtuoso Technology Data Constraint 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 Constraint Reference

### Spacing Constraints (One Layer)

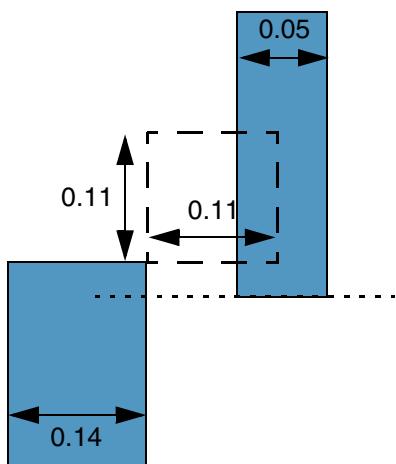
---

The minimum spacing requirement for a concave corner is 0.15, except when a concave edge is less than 0.05 in length.

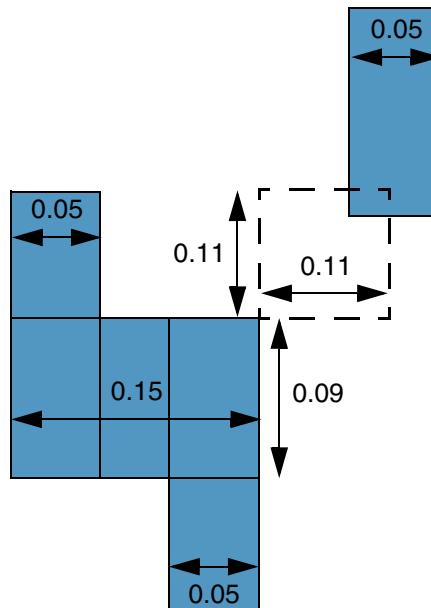
```

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

```



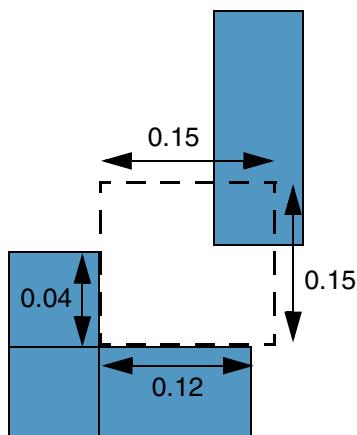
a) The constraint does not apply. The parallel run length between the convex corner and the shape on the right is greater than zero.



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 Constraint 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 Constraint Reference

### Spacing Constraints (One Layer)

---

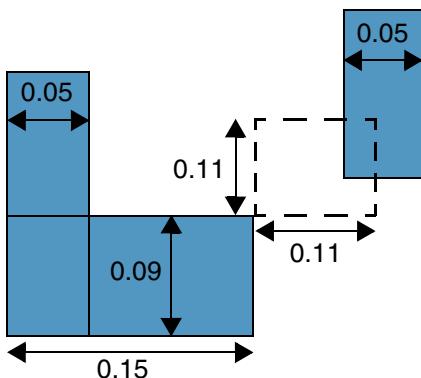
#### ***Example 3: minCornerSpacing with convex, exceptEndOfLineWidth, exceptJogLength, and treatAsJog***

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.

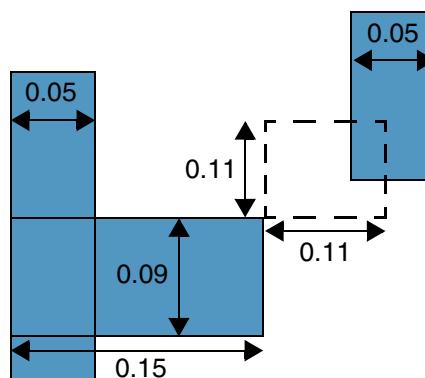
```

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

```



- 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 Constraint Reference

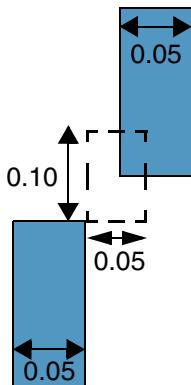
### Spacing Constraints (One Layer)

#### **Example 4: minCornerSpacing with convex and withinCornerOnly**

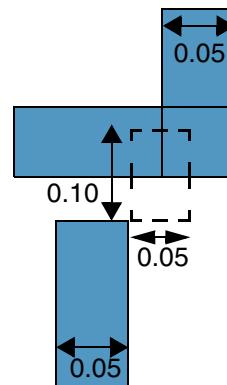
The minimum spacing between a convex corner and a neighboring corner is 0.10 in the vertical direction and 0.05 in the horizontal direction.

```
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.

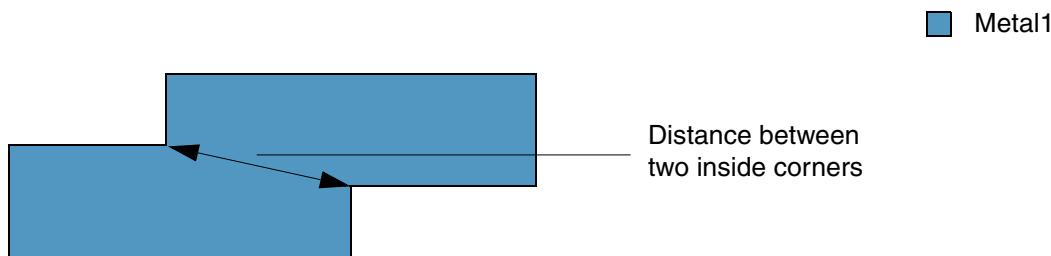


b) The constraint does not apply. No corner is found inside the search window.

## **minCornerToCornerDistance**

```
spacings(  
  ( minCornerToCornerDistance tx_layer  
    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_spacing</i>	The distance between the inside corners of a shape must be greater than or equal to this value.

### **Parameters**

None

## minDiagonalSpacing

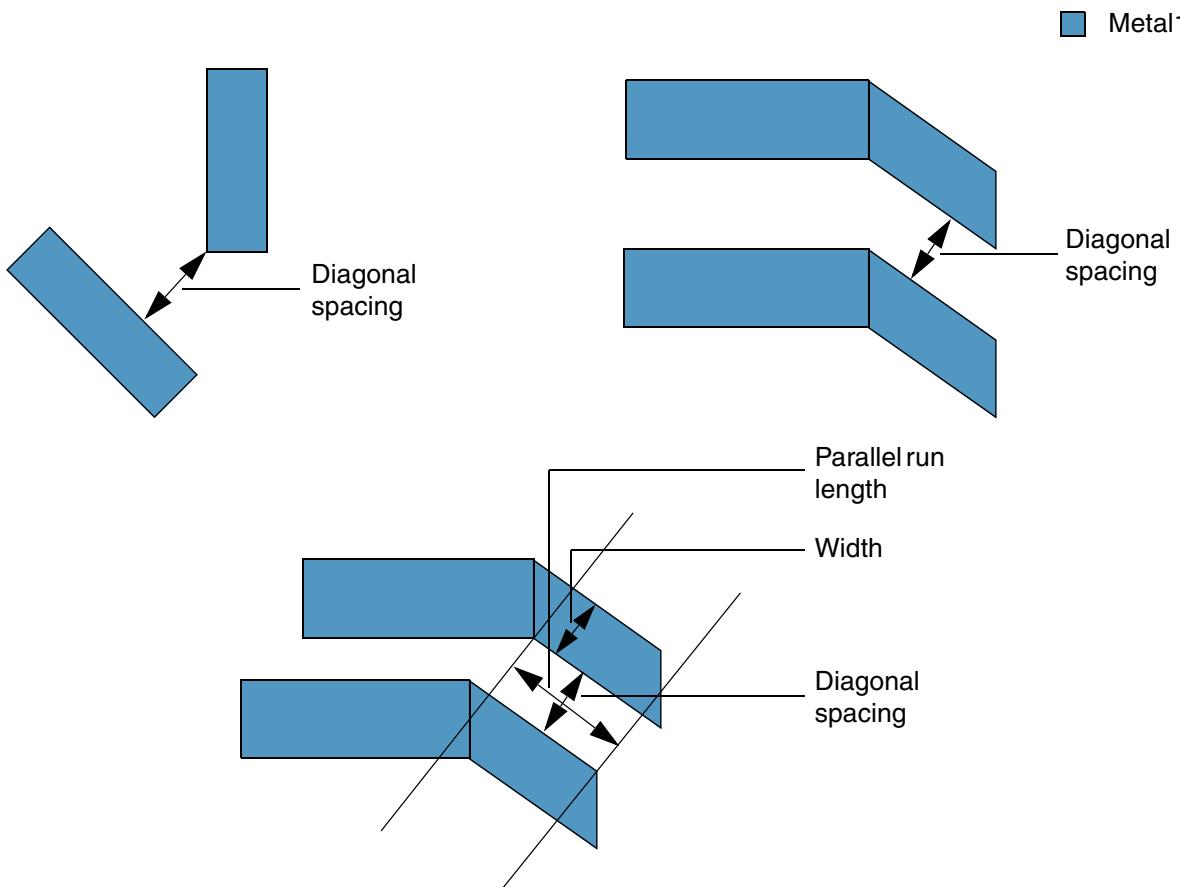
```

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 actual or Euclidean diagonal distance, and not as the Manhattan 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 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</i> ) <i>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 indices.  <b>Note:</b> 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.
------------------	---

## 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 `mindiagspace1`.

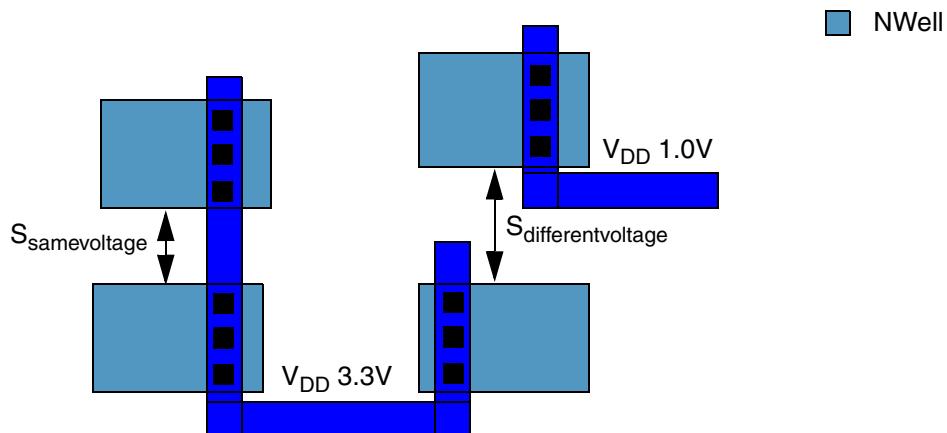
```
spacings(
  ( minDiagonalSpacing "Poly1"
    0.7
  )
  ( minDiagonalSpacing "Metal2"
    techParam("mindiagspace1")
  )
) ;spacings
```

## 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

<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 shapes that are at different voltage levels must be greater than or equal to this value.

## Parameters

None

## 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
```

### **minEdgeLengthSpacing (Advanced Nodes Only)**

```

spacings(
    ( minEdgeLengthSpacing tx_layer
        'length f_length ['prl f_parallelRunLength]
        ['horizontalEdge | 'verticalEdge]
        f_spacing
    )
) ;spacings

```

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

*tx\_layer* The layer on which the constraint is applied.

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

*f\_spacing* The minimum required spacing between shapes.

## Parameters

'length *f\_length*

The constraint applies only to the edges with length less than this value.

'prl *f\_parallelRunLength*

The constraint applies only if the parallel run length between the two edges is greater than or equal to this value.

'horizontalEdge | 'verticalEdge

The constraint applies only to the edges in the specified direction. The default is any direction.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

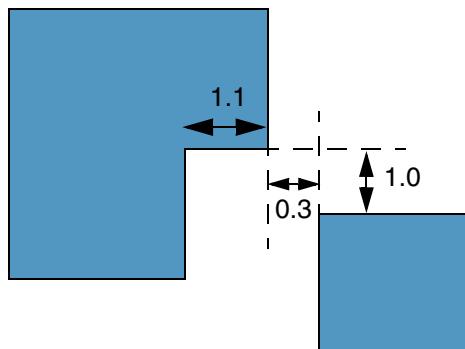
### Spacing Constraints (One Layer)

#### Example

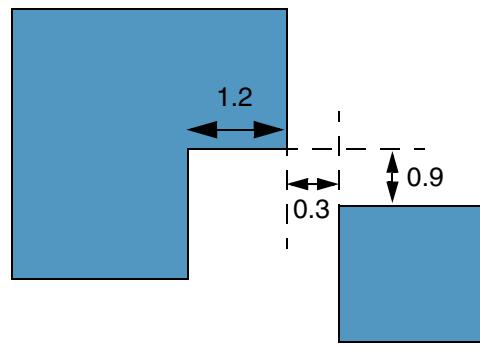
The spacing between a shape and the horizontal short edge of another shape on Metal1 must be at least 1.0 if the short edge is less than 1.2 long and the parallel run length between the two edges is greater than or equal to -0.3.

```
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.

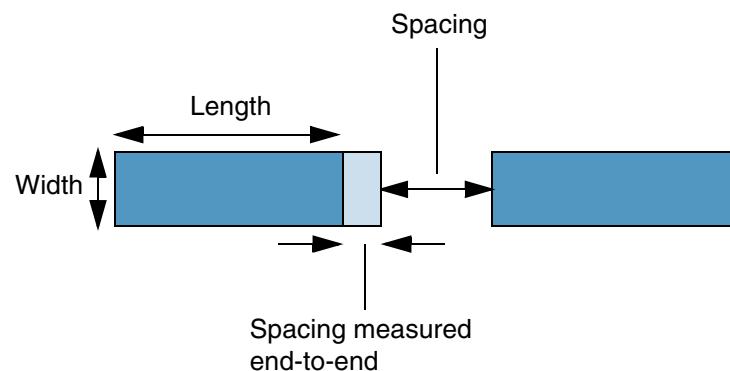
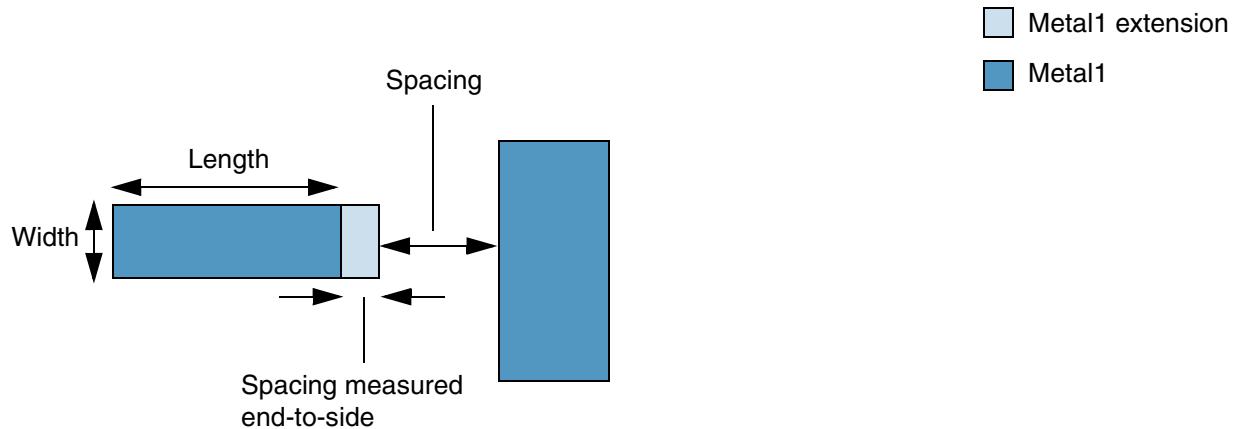
## minEndOfLineExtensionSpacing

```

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.



## 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

'length <i>f_length</i>	The constraint applies only if the length is greater than or equal to this value.
'twosides	The constraint applies only if the length of both sides is greater than or equal to <i>length</i> . Type: Boolean
'sameMask	(Advanced Nodes Only) The constraint applies only to same-mask shapes. Type: Boolean
'negativePRL	(Advanced Nodes Only) 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
'nonEol	(ICADV12.3 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
'otherWidth <i>f_otherWidth</i>	(ICADV12.3 Only) The constraint applies only if the width of the neighboring shape is less than this value.

```
'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 actual 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 Constraint Reference

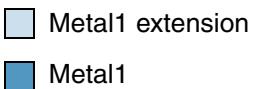
### Spacing Constraints (One Layer)

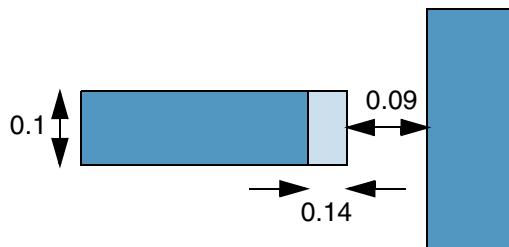
---

#### ***Example 1: minEndOfLineExtensionSpacing***

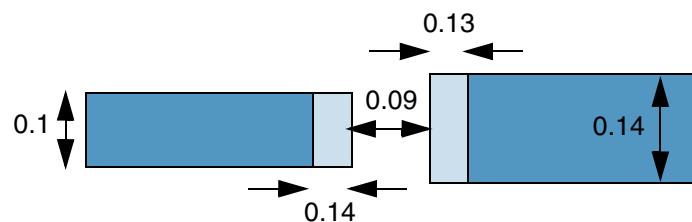
An end-of-line edge with width less than 0.11 requires 0.14 extension, and an end-of-line edge with width greater than or equal to 0.11 and less than 0.15 requires 0.12 extension when spacing is measured end-to-side and 0.13 extension when spacing is measured end-to-end. The edge-to-edge spacing must 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
```

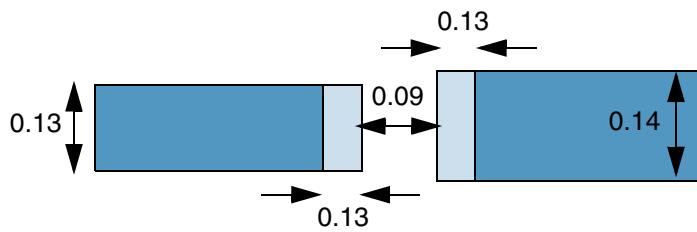

  
 Metal1 extension
   
 Metal1



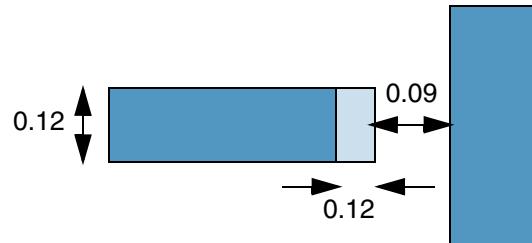
a) FAIL. The spacing measured after applying 0.14 extension is 0.09 (<0.1).



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 Constraint Reference

### Spacing Constraints (One Layer)

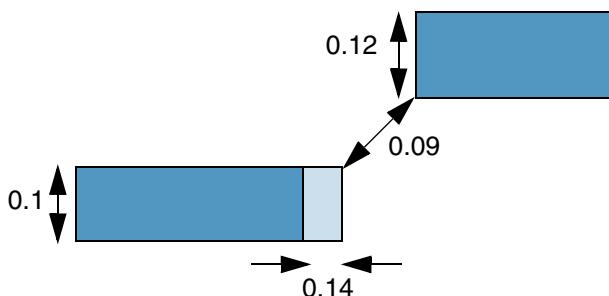
---

#### ***Example 2: minEndOfLineExtensionSpacing with negativePRL***

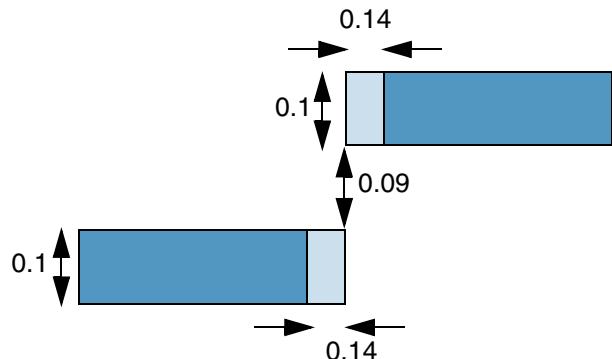
The minimum spacing between an end-of-line edge and the neighboring edges with which it has a negative parallel run length must be 0.1. If the end-of-line edge is less than 0.11 wide, a 0.14 extension is applied before spacing is checked.

```
spacings(
  ( minEndOfLineExtensionSpacing "Metall1"
    '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 Constraint Reference

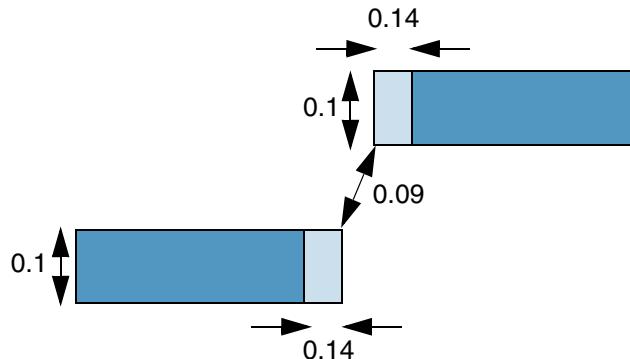
### Spacing Constraints (One Layer)

#### **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



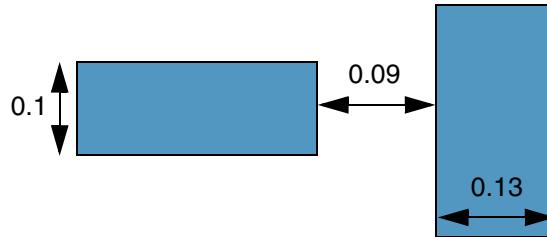
The constraint does not apply. The neighboring edge that has a parallel run length less than zero with the end-of-line edge is also an end-of-line edge. The constraint would fail if 'nonEol' was not specified.

**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



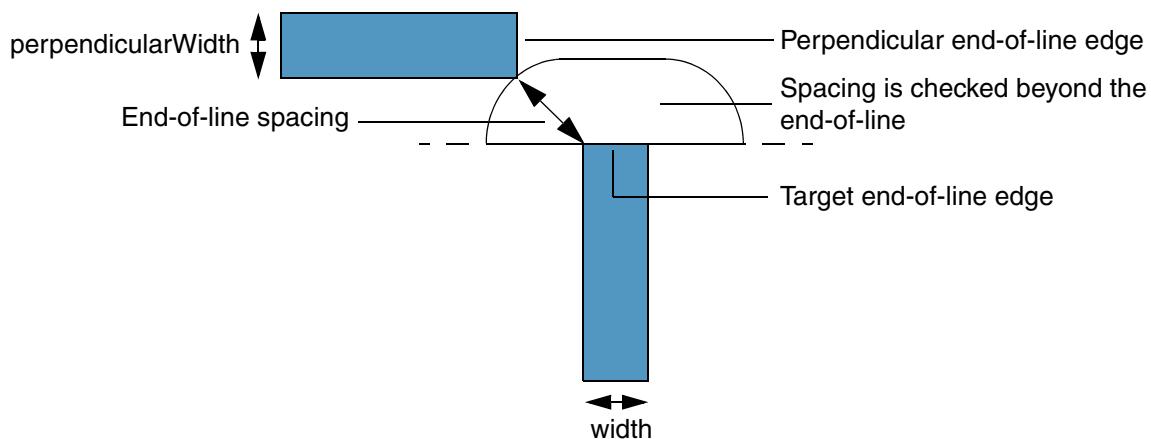
The constraint does not apply because the width of the neighboring wire is 0.13 (>0.12). A violation would occur if '`'otherWidth`' was not specified.

## **minEndOfLinePerpSpacing**

```
spacings(
  ( minEndOfLinePerpSpacing tx_layer
    'width f_width
    'perpWidth f_perpendicularWidth
    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**

*tx\_layer*      The layer on which the constraint is applied.

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

*f\_spacing*      The minimum required spacing between the two edges.

## Virtuoso Technology Data Constraint 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*

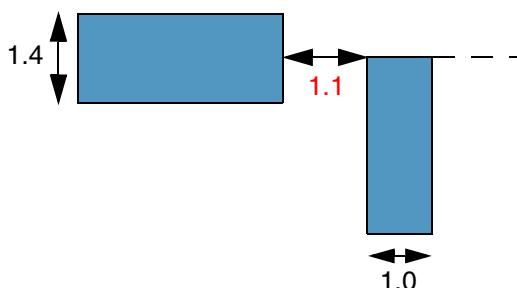
The constraint applies only if the width of the perpendicular edge is less than this value.

#### Example

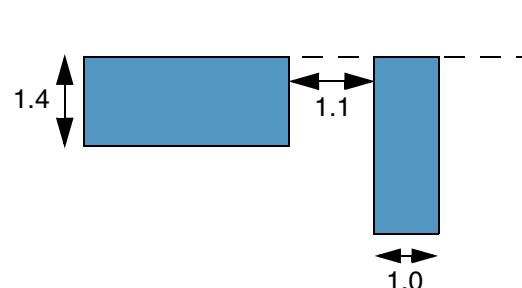
The spacing between an end-of-line edge and a perpendicular edge that lies beyond the end-of-line edge on layer Metal1 must be at least 1.2 if the width of the end-of-line edge is less than 1.1 and the width of the perpendicular edge is less than 1.5.

```
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.

## minEndOfLineSpacing

```
spacings(
  ( minEndOfLineSpacing tx_layer
    ['horizontal | 'vertical]
    'width f_width ['oppWidth f_oppWidth]
    'distance f_eolWithin
    ['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)}]
    ]
    [[['maxLength f_maxLength | 'minLength f_minLength ['twoSides]]
      ['bothWires]
    ]
    ['equalRectWidth]
    ['encloseDistance f_encloseDist 'cutToMetalSpace f_cutToMetalSpace
      ['above | 'below]
      ['allCuts]
    ]
    ['exceptExactAligned]
    ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
      | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
    ]
    f_spacing
  )
)
```

## Virtuoso Technology Data Constraint Reference

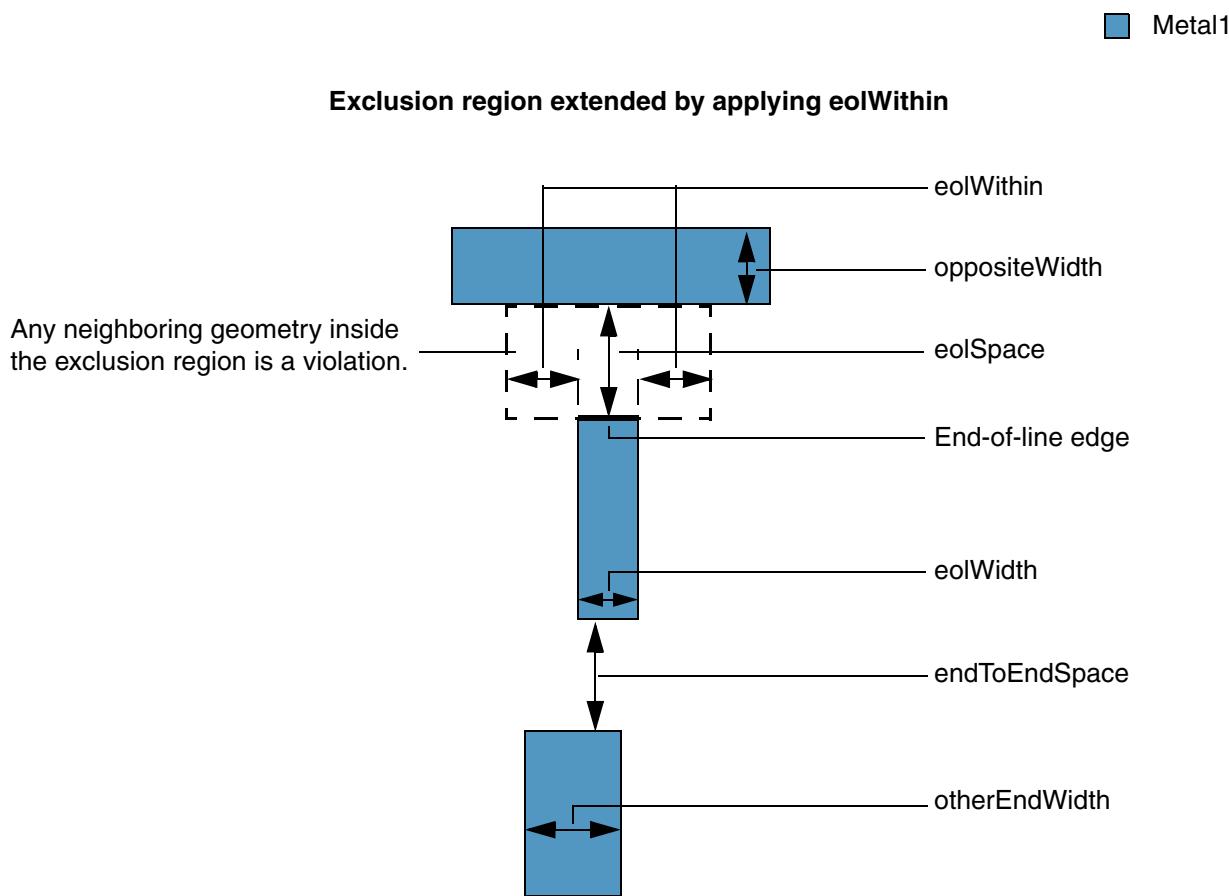
### Spacing Constraints (One Layer)

) ;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*.

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 on either side of the end-of-line shape. Edge-to-edge spacing must be greater than or equal to *eolSpace*.

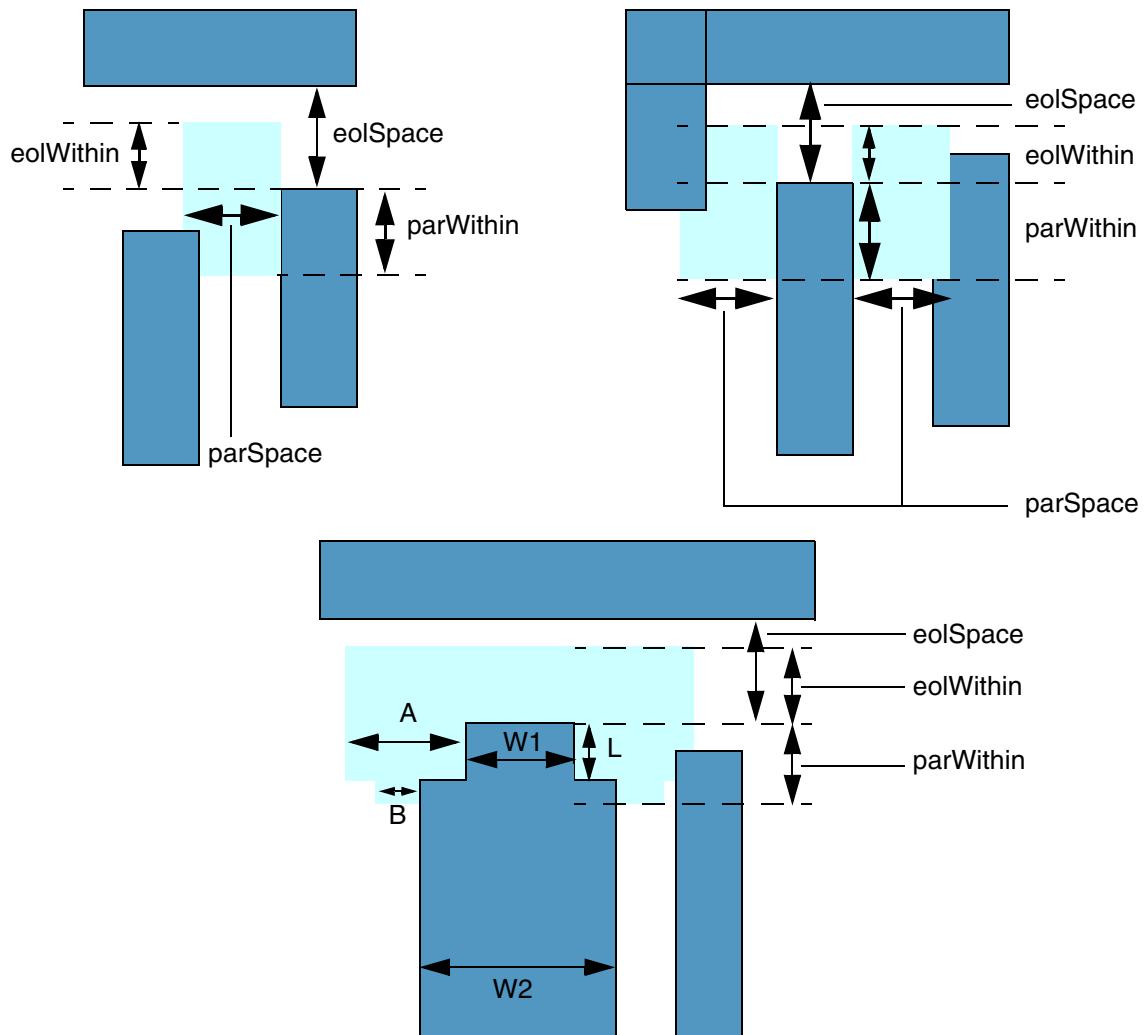
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

Metal1  
 Search window

**Constraint applies only if the specified number of parallel edges are present inside the search window**



Assuming '`subtractWidth`' is specified in this case,  $A = \text{parSpace} - W_1$ , and  $B = \text{parSpace} - W_2$ , 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.

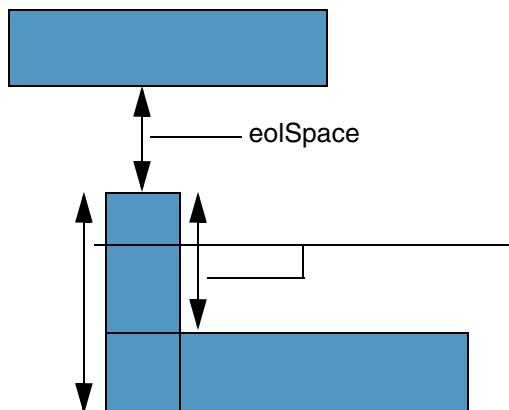
## Virtuoso Technology Data Constraint Reference

### 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*.

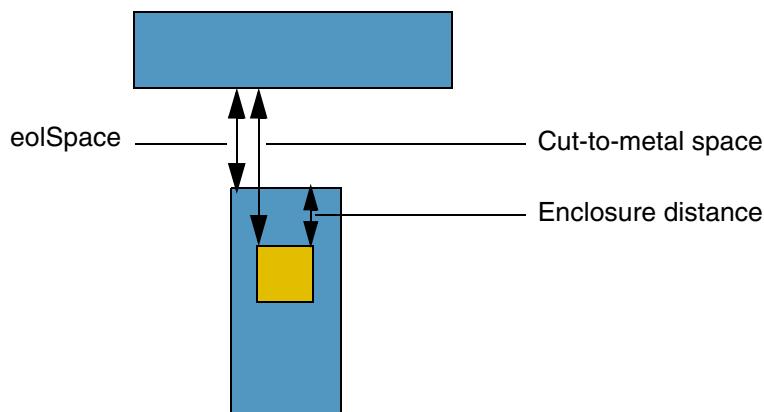
- █ Via1
- █ Metal1

**Constraint applies only if the minimum and maximum length requirement is satisfied**



The constraint does not apply if both lengths are greater than *maxLength* or less than *minLength*. If 'twoSides' is specified, the constraint applies only if both lengths are greater than or equal to *minLength*.

**Constraint applies only if cut-to-metal and cut-to-enclosure-edge spacing requirement is satisfied**



## Virtuoso Technology Data Constraint Reference

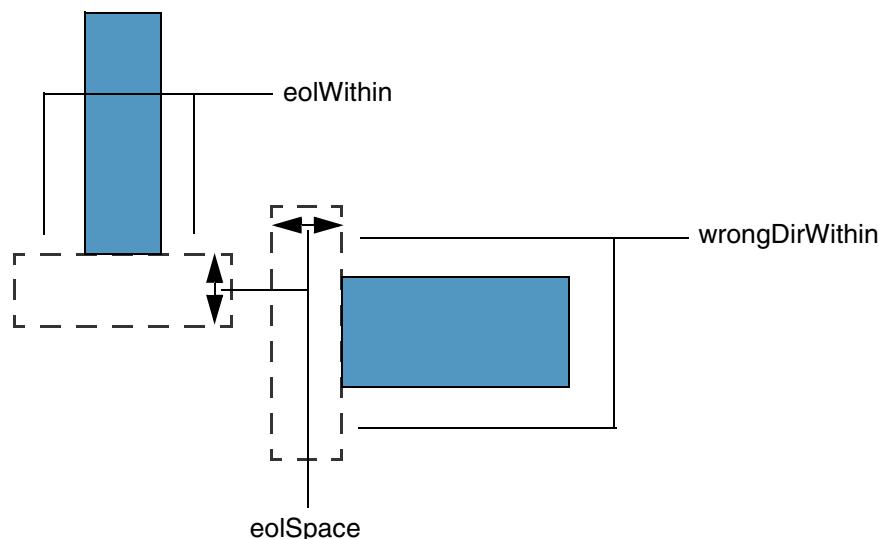
### Spacing Constraints (One Layer)

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.

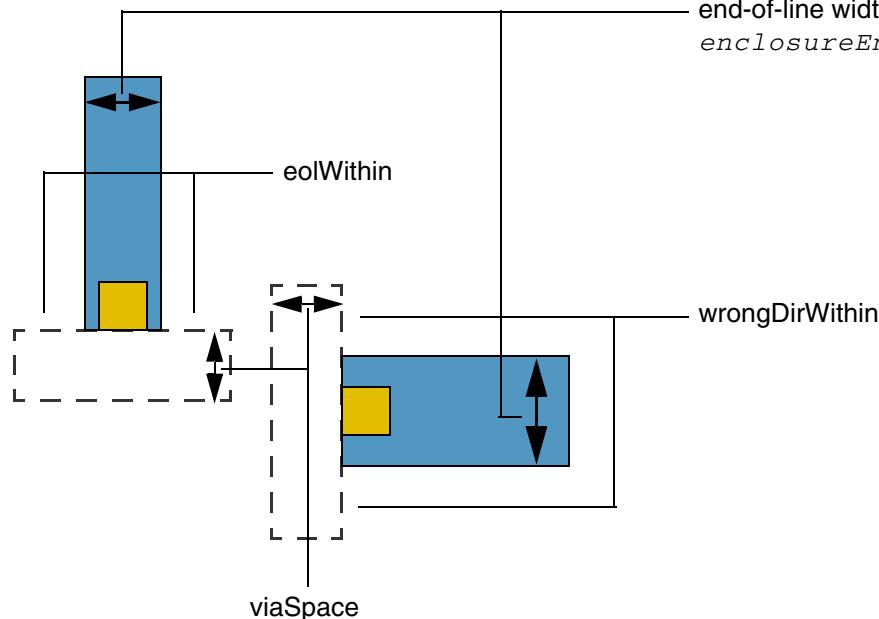
↑ The preferred routing direction is vertical and any neighboring geometry inside the exclusion region (depicted using dashed lines) is a violation.

■ Via1  
■ Metal1

**Exclusion region extended by applying *eolWithin* and *wrongDirWithin***



The constraint applies only if the end-of-line width is less than *enclosureEndWidth*.

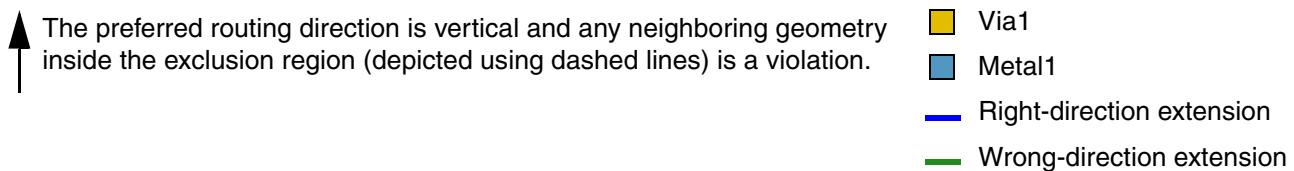


## Virtuoso Technology Data Constraint Reference

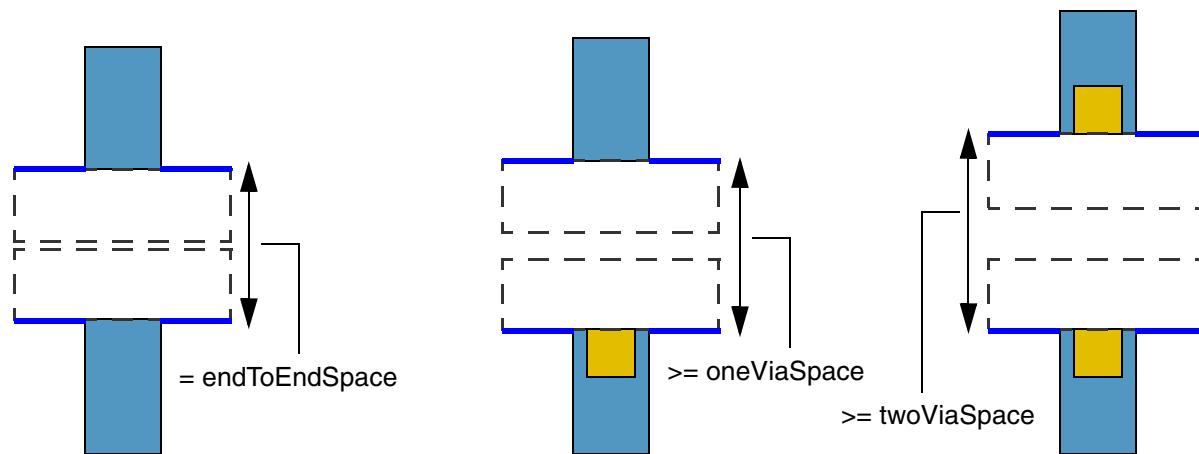
### Spacing Constraints (One Layer)

---

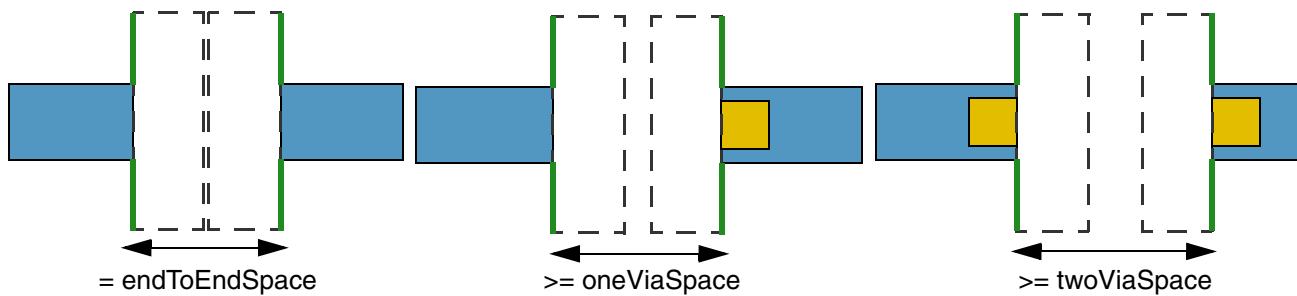
In the following figures, you can see the right-direction and wrong-direction extensions applied to two shapes that are face-to-face. The spacing requirement varies based on the number of vias present.



**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.

## 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_spacing</i>	The minimum required spacing between an end-of-line edge and its neighboring geometry.

## Parameters

'horizontal | 'vertical

(Advanced Nodes Only) The direction in which the constraint applies. If direction is not specified, the constraint applies to all end-of-line edges.

- 'horizontal: The constraint applies only to vertical end-of-line edges.
- 'vertical: The constraint applies only to horizontal end-of-line edges.

Type: Boolean

'width *f\_eolWidth*

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

'oppWidth *f\_oppWidth*

The constraint applies only if a neighboring shape perpendicular to the end-of-line edge has width equal to this value.

'distance *f\_eolWithin*

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).

## Virtuoso Technology Data Constraint 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 Constraint Reference

### Spacing Constraints (One Layer)

---

'sameNet | 'sameMetal

(Advanced Nodes Only) 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

(ICADV12.3 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 Constraint 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 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

'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*)

(Advanced Nodes Only) 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*

(Advanced Nodes Only) 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*

(Advanced Nodes Only) An end-of-line edge is extended on both sides by this value before the constraint is applied.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'sameMask | 'diffMask

(Advanced Nodes Only) 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*

(ICADV12.3 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

(Advanced Nodes Only) The constraint applies only if the end-of-line width is equal to this value.

Type: Boolean

'exceptExactEolWidth (*f\_exactWidth f\_otherWidth*)

(ICADV12.3 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*

(Advanced Nodes Only) 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 Constraint 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 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.

'oneCutSpace *f\_oneCutSpace*

The spacing between the two end-of-line edges, one of which touches a via cut, must be equal to this value.

'twoCutSpace *f\_twoCutSpace*

The spacing between the two end-of-line edges, both of which touch a via cut each, must be equal to this value.

'extension {*f\_extension* | (*f\_extension* *f\_wrongDirExtension*)}

- *f\_extension*: The extension applied to both sides 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 edge and horizontal extensions (right-direction extensions).
- *f\_wrongDirExtension*: The extension applied to both sides 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 edge and vertical extensions (wrong-direction extensions).

'maxLength *f\_maxLength*

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

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'bothWires	(Advanced Nodes Only) The spacing between two end-of-line shapes must be greater than or equal to <code>endToEndSpace</code> if the length of both shapes is greater than or equal to <code>minLength</code> and less than or equal to <code>maxLength</code> .  This parameter must be specified only if ' <code>endtoEndSpace</code> and one of ' <code>maxLength</code> or ' <code>minLength</code> 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 <code>minEndOfLineSpacing</code> constraints are defined on a layer and ' <code>equalRectWidth</code> is specified with even one of them, then ' <code>equalRectWidth</code> must be specified with all of them.
	Type: Boolean
'encloseDistance <code>f_encloseDistance</code>	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 <code>f_cutToMetalSpace</code>	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
'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	(Advanced Nodes Only) The constraint applies only if the shapes are not exactly aligned.
	Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

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

(ICADV12.3 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 (*tx\_layer1 tx\_layer2 ... tx\_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 (*tx\_layer1 tx\_layer2 ... tx\_layerN*).

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

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

### 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](#)

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

- [Example 12: minEndOfLineSpacing with equalRectWidth](#)
- [Example 13: minEndOfLineSpacing with cutEolSpace, wrongDirWithin, and endToEndSpace](#)
- [Example 14: minEndOfLineSpacing with horizontal](#)
- [Example 15: minEndOfLineSpacing with paraEdgeWithin, paraEdgeSpace, minLength, encloseDistance, and cutToMetalSpace](#)

## Virtuoso Technology Data Constraint Reference

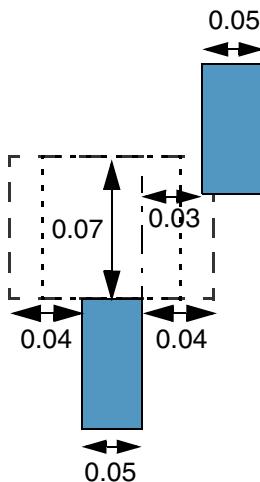
### Spacing Constraints (One Layer)

#### **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 exactly 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 Constraint Reference

### Spacing Constraints (One Layer)

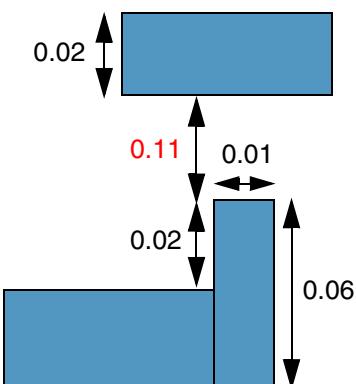
---

#### **Example 2: minEndOfLineSpacing with oppWidth and minLength**

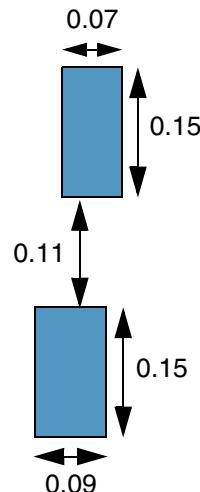
The end-of-line spacing between Metal1 wires must be at least 0.12 when the minimum length of an end-of-line side is less than 0.06 and the width of the wire perpendicular to the end-of-line edge is 0.08. The lateral verification distance for the spacing check is 0.05.

```
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.

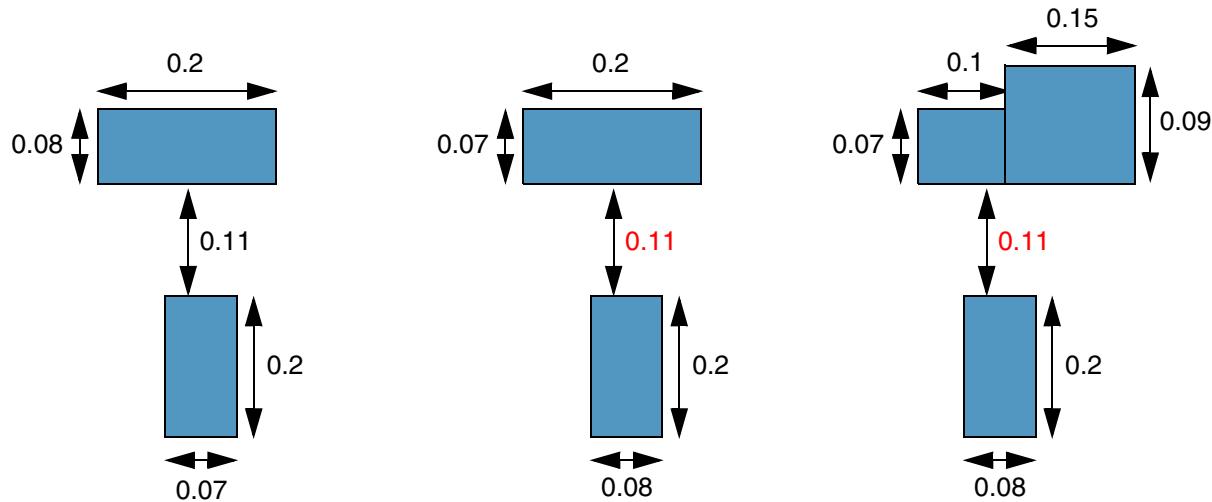


b) The constraint does not apply because the opposite wire has a perpendicular span of 0.15 ( $>0.08$ ) to the end-of-line edge.

## Virtuoso Technology Data Constraint 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 Constraint Reference

### Spacing Constraints (One Layer)

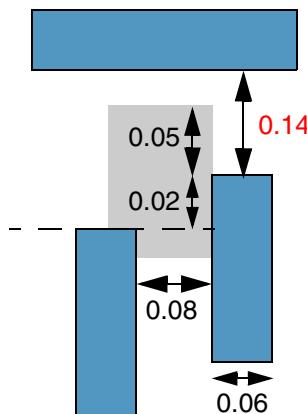
---

```

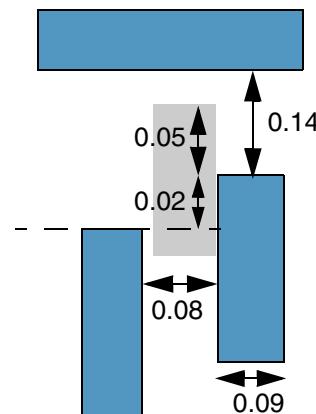
spacings(
    minEndOfLineSpacing "Metall1"
        'width 0.1
        'distance 0.05
        'paraEdgeWithin 0.03 'paraEdgeSpace 0.16
        'subtractWidth
        0.15
    )
) ;spacings

```

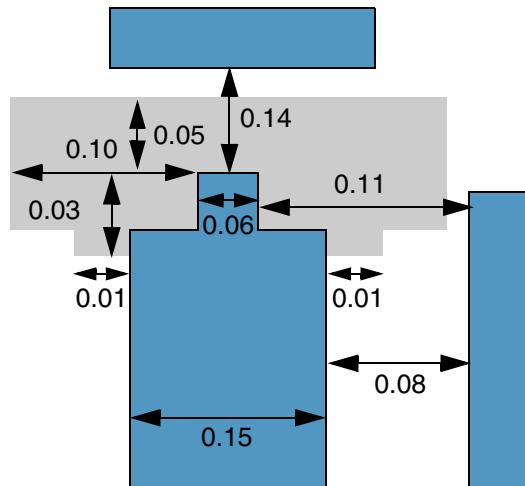
 Metal1  
 Search window



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 Constraint Reference

### Spacing Constraints (One Layer)

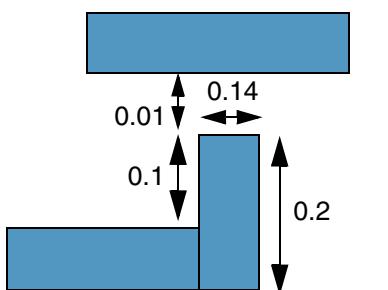
---

#### **Example 4: minEndOfLineSpacing with endToEndSpace, minLength, and twoSides**

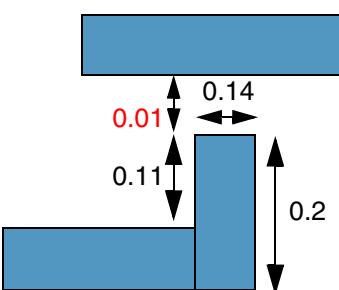
The end-of-line spacing between Metal1 wires must be at least 0.1 and the end-to-end spacing must be 0.12 if the length of both end-of-line sides is greater than or equal to is 0.11.

```
spacings(
  ( minEndOfLineSpacing "Metal1"
    'width 0.15
    'distance 0.05
    'endToEndSpace 0.12
    'minLength 0.11 'twoSides
    0.1
  ) ;spacings
```

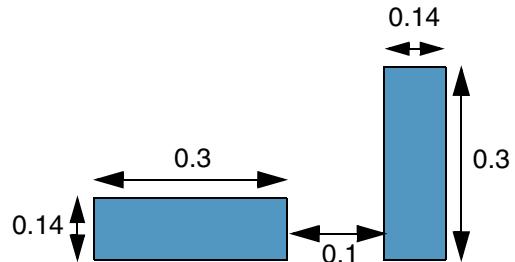
  Metal1



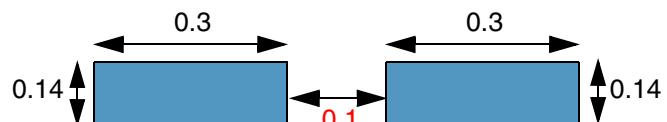
a) The constraint does not apply because only the right end-of-line side is longer than 0.11. A violation would occur if 'twoSides' was not specified.



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 0.1 (<0.12).

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

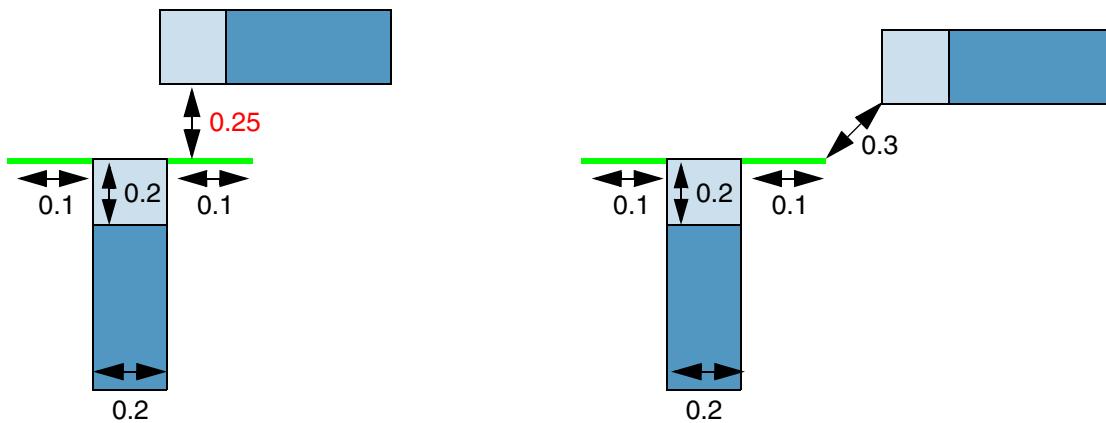
---

#### **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
        'widthRanges ("<=0.3" 0.6 0.8)
        'extendBy 0.2
        'sizeBy 0.1
        0.3
    )
) ;spacings
```

	<b>Metal1</b> <b>Metal1 extension</b> <b>Sized by</b>
---	---



a) FAIL. The width of the end-of-line (0.2) is within a specified range ( $\leq 0.3$ ), but the end-of-line spacing is 0.25 ( $< 0.3$ ).

b) PASS. The width of the end-of-line (0.2) is within a specified range ( $\leq 0.3$ ) and the end-of-line spacing is 0.3.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

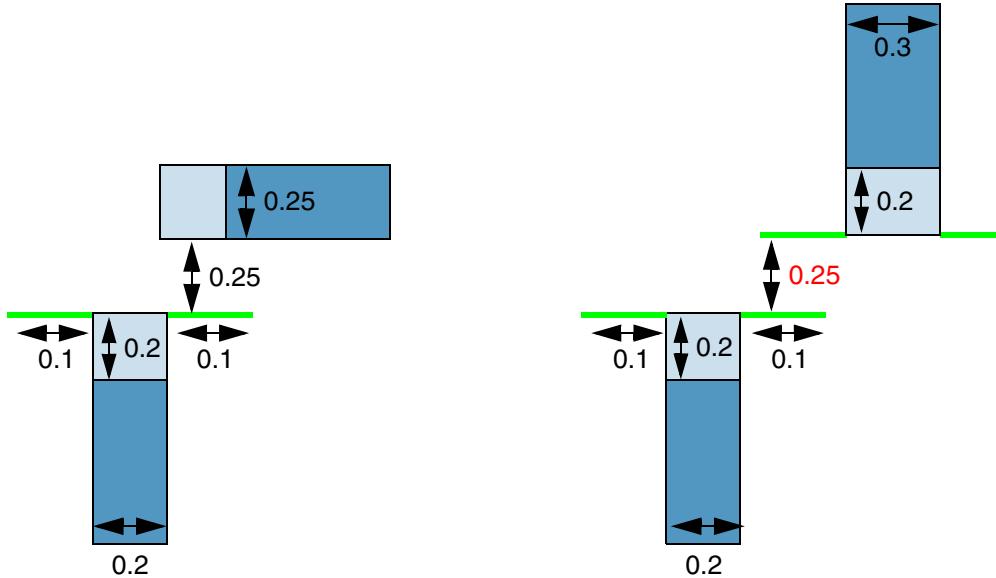
---

#### **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"
    'widthRanges ("<=0.3" "0.6" "0.8")
    'extendBy 0.2
    'sizeBy 0.1
    'endToEndSpace 0.3
    0.3
  )
) ;spacings
```

<span style="background-color: #4682B4; border: 1px solid black; padding: 2px;"></span> Metal1 <span style="background-color: #ADD8E6; border: 1px solid black; padding: 2px;"></span> Metal1 extension <span style="color: green;">—</span> Sized by
---



a) The constraint does not apply because the end-of-line edges do not face each other (the width of both edges is less than 0.3).

b) FAIL. The two end-of-line edges (width  $\leq 0.3$ ) 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 Constraint Reference

### Spacing Constraints (One Layer)

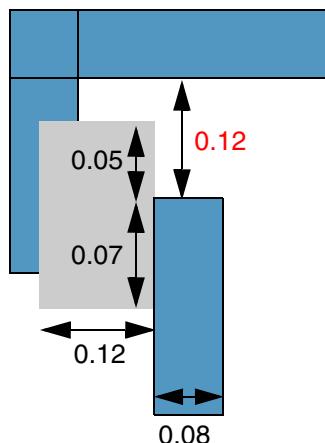
---

#### **Example 7: minEndOfLineSpacing with paraEdgeWithin, paraEdgeSpace, and sameMetal**

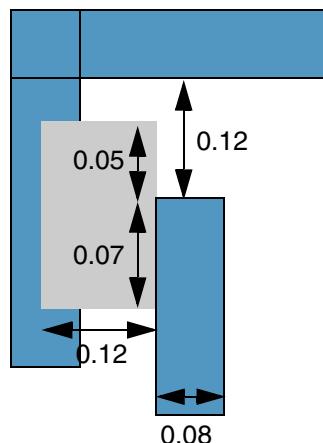
The end-of-line spacing between Metal1 wires must be at least 0.13 if a parallel edge present inside the search window is part of a contiguous same-metal shape. The search window is defined by `eolWithin = 0.05`, `parWithin = 0.07`, and `parSpace = 0.12`.

```
spacings(
  ( minEndOfLineSpacing "Metal1"
    'width 0.1
    'distance 0.05
    'paraEdgeWithin 0.07 'paraEdgeSpace 0.12
    'sameMetal
    0.13
  ) ;spacings
```

Metal1  
 Search window



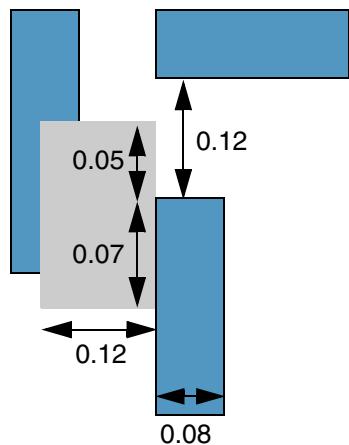
a) FAIL. The neighboring edges on the left and on top belong to a contiguous same-metal shape. However, the end-of-line spacing is 0.12 (<0.13).



b) The constraint does not apply because the neighboring left edge covers the entire length of the search window.

## Virtuoso Technology Data Constraint 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 Constraint 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 equal to 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
```

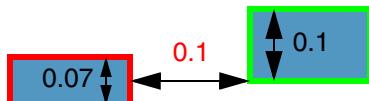
- █ Metal1
- █ mask1
- █ mask2



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 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 0.1 (<0.13).

## Virtuoso Technology Data Constraint Reference

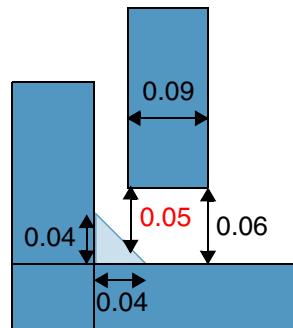
### Spacing Constraints (One Layer)

#### **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).

**Example 10: *minEndOfLineSpacing* with *exceptExactEoWidth***

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
    'exceptExactEoWidth (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 *nonEoICornerOnly***

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 Constraint Reference

### Spacing Constraints (One Layer)

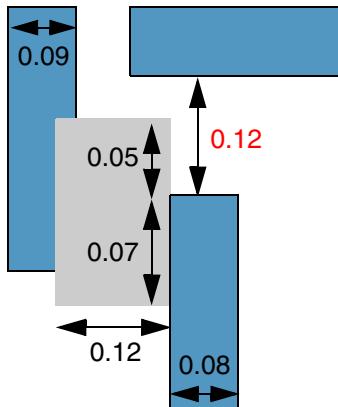
---

```

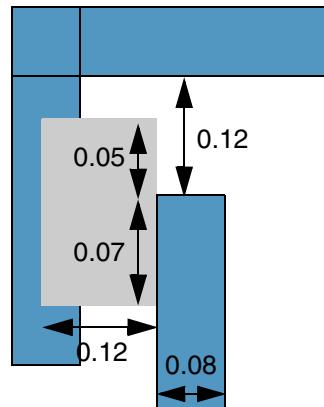
spacings(
    ( minEndOfLineSpacing "Metall1"
        'width 0.1
        'distance 0.05
        'paraEdgeWithin 0.07 'paraEdgeSpace 0.12
        'nonEoLCornerOnly
        0.13
    )
) ; spacings

```

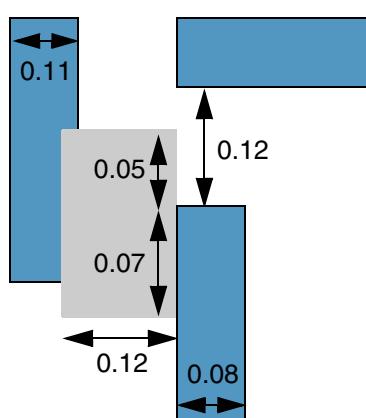
Metal1  
 Search window



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.



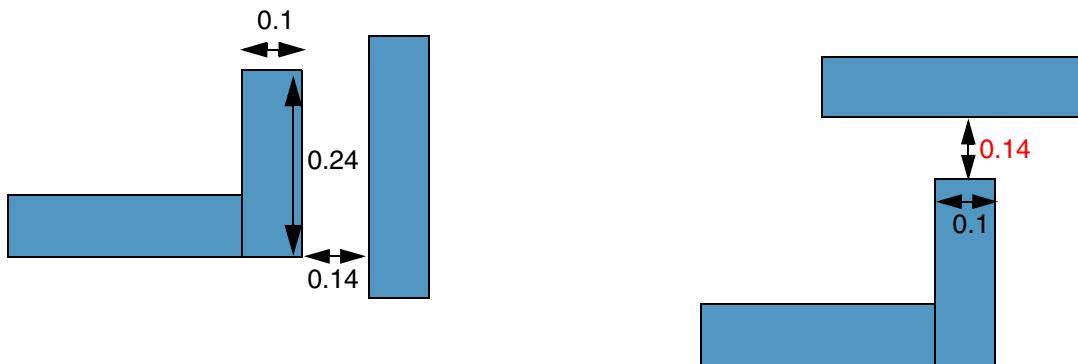
c) The constraint does not apply because the corner inside the search window is not formed by an end-of-line edge (the end-of-line width of the neighboring shape is 0.11, which is greater than 0.1).

**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 *cutEoISpace*, *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 equal 0.11 if the end-of-line edges do not touch a via cut, greater than 0.13 if an end-of-line edge touches a via cut, and greater than 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 Constraint Reference

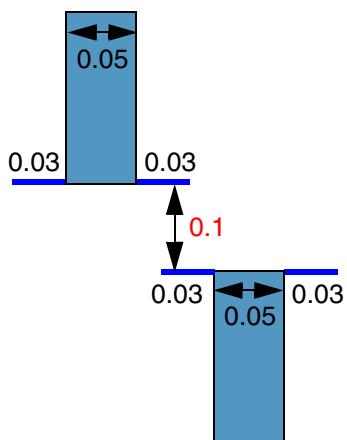
## Spacing Constraints (One Layer)

```

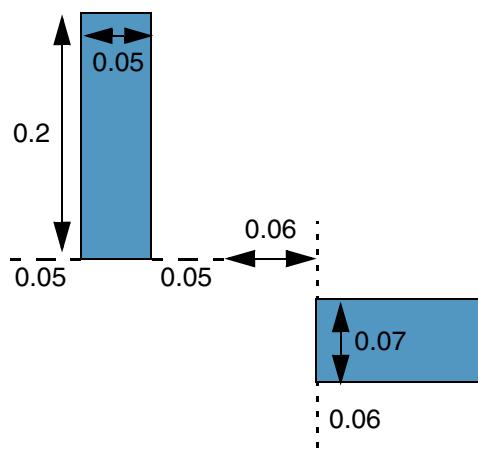
spacings(
    minEndOfLineSpacing "Metall1"
        width 0.08
        distance 0.05
        cutEolSpace 0.12
            cutClass "VA"
            enclosureEndWidth 0.14
                enclosureEndWithin 0.05
        wrongDirWithin 0.06
        endToEndSpace 0.11
            oneCutSpace 0.13 twoCutSpace 0.15
            extension 0.03 0.08
    0.1
)
) ;spacings

```

- █ Via1
- Metal1
- Right direction extension
- ↑ Preferred routing direction - Vertical



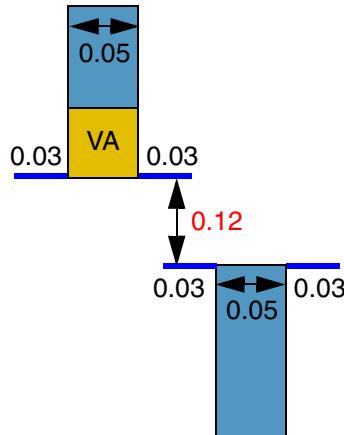
a) FAIL. The face-to-face edges are both less than 0.08 wide and have a parallel run length greater than zero after an extension of 0.03 is applied to the edges in the non-preferred routing direction. However, the end-to-end spacing is 0.1 (<0.11).



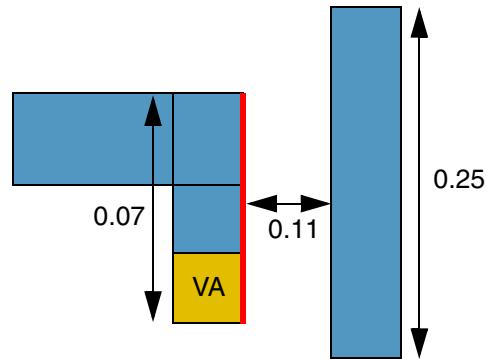
b) PASS. The end-of-line spacing is 0.11 (0.05+0.06), which is greater than the required value of 0.1. (The dashed lines indicate the lateral verification distance in the non-preferred routing direction (*eolWithin*) and the dotted lines indicate the lateral verification distance in the preferred routing direction (*wrongDirWithin*)).

## Virtuoso Technology Data Constraint 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.)

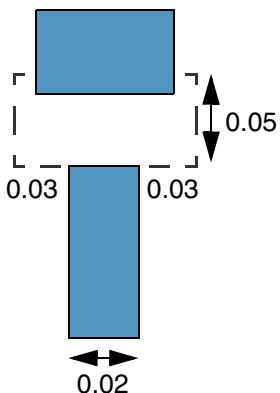
**Example 14: minEndOfLineSpacing with horizontal**

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:

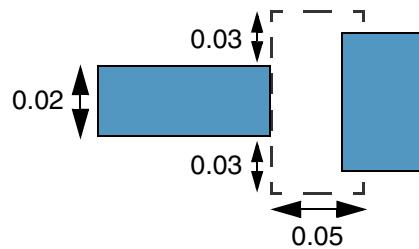
- The end-of-line width is less than 0.03.
- The lateral verification distance for the spacing check is 0.03.

```
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).

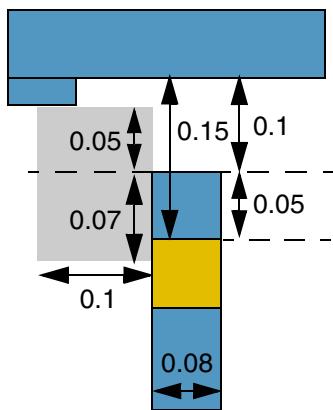
**Example 15: minEndOfLineSpacing with paraEdgeWithin, paraEdgeSpace, minLength, encloseDistance, and cutToMetalSpace**

The end-of-line spacing between Metal1 wires must be at least 0.15 if the following conditions are met:

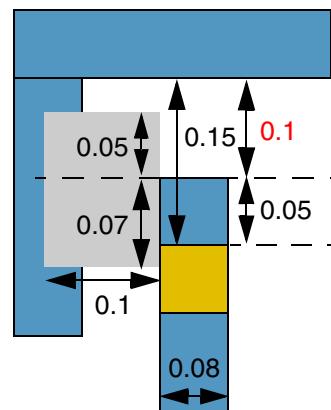
- The width of the end-of-line shape is less than 0.1 (*eolWidth*) and the length of the end-of-line side is greater than or equal to 0.1 (*paraMinLength*).
- A parallel edge is present inside the search window defined by *eolWithin* = 0.05, *parWithin* = 0.07, and *parSpace* = 0.1.
- The distance of 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
```





a) The constraint does not apply because no parallel edge is present inside the search window.

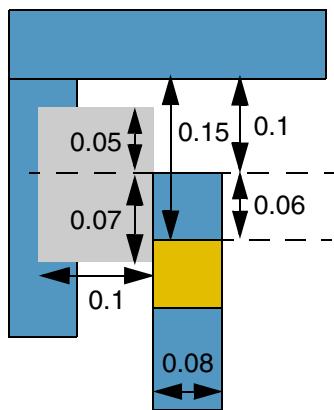


b) FAIL. The enclosed distance is 0.05 (<0.06) and the cut-to-metal space is 0.15 (<0.16). Additionally, the neighboring edge meets the criteria specified for a parallel edge. However, the end-of-line spacing is 0.01 (<0.15).

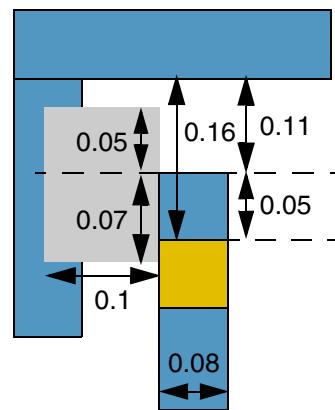
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

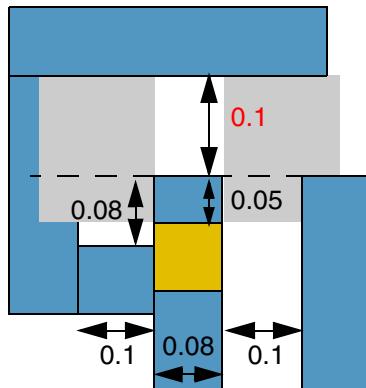
---



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.

## **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> )	(Advanced Nodes Only) The constraint applies only if the edges forming the concave corner satisfy the following conditions: <ul style="list-style-type: none"><li>■ Any one edge is greater than the first value</li><li>■ The other edge is greater than the second value.</li></ul>

## Examples

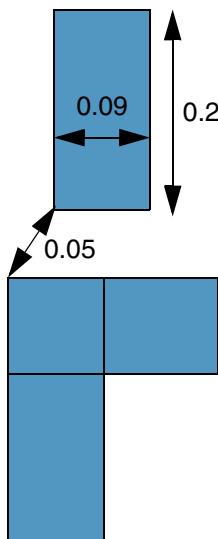
- [Example 1: minEndOfLineToConcaveCornerSpacing with width and minLength](#)
- [Example 2: minEndOfLineToConcaveCornerSpacing with width and dualAdjacentLength](#)

### ***Example 1: minEndOfLineToConcaveCornerSpacing with width and minLength***

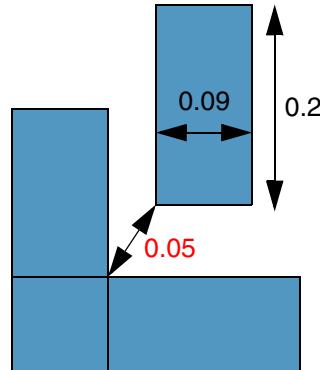
The minimum spacing between an end-of-line edge less than or equal to 0.1 wide and a concave corner must be greater than or equal to 0.06 if both adjoining edges of the end-of-line edge are greater than or equal to 0.2 long.

```
spacings(
  ( minEndOfLineToConcaveCornerSpacing "Metall1"
    '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 Constraint Reference

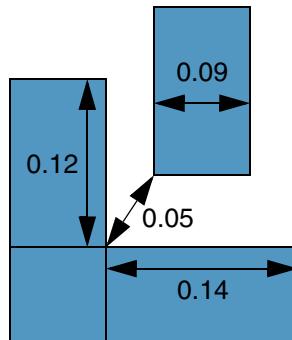
### Spacing Constraints (One Layer)

#### ***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 (Advanced Nodes Only)**

```
spacings(  
  ( minEndOfLineToNotchSpacing tx_layer  
    'width f_width 'notchLength f_notchLength  
    f_spacing  
  )  
) ;spacings
```

Specifies the minimum spacing between an end-of-line edge and a notch with the specified length.

### **Values**

*tx\_layer*                   The layer on which the constraint is applied.

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

*f\_spacing*               The required minimum spacing.

### **Parameters**

'width *f\_width*

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

'notchLength *f\_notchLength*

The constraint applies only if the length of the notch is less than this value.

## Virtuoso Technology Data Constraint Reference

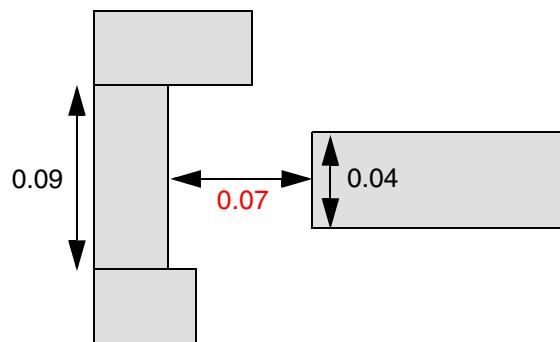
### Spacing Constraints (One Layer)

#### 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



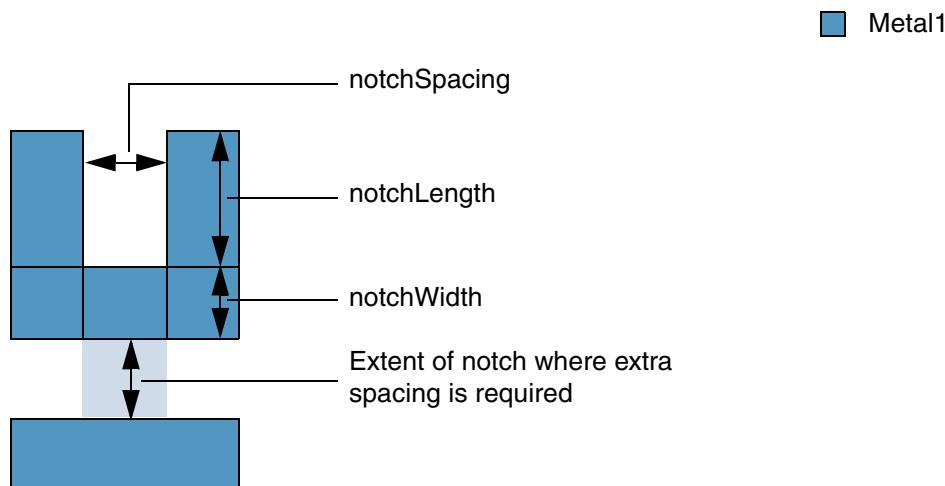
FAIL. The end-of-line width is 0.04 (<0.05) and the length of the notch is 0.09 (<0.1), but the distance of the end-of-line edge from the notch is 0.07 (<0.08).

## **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 Constraint 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*

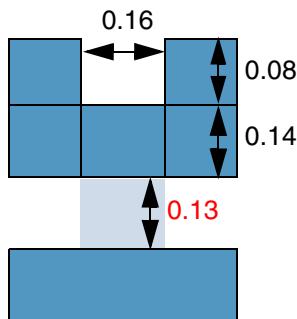
The constraint applies only if notch spacing is less than or equal to this value.

#### Example

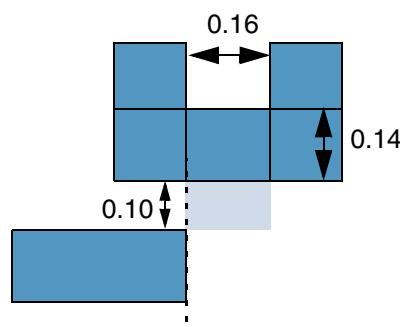
The spacing between the closed end of a notch and a shape that has a parallel overlap with the notch must be at least 0.14 if all of the following conditions are satisfied: notch width is less than 0.15, notch length is greater than or equal to 0.08, and notch spacing is less than or equal to 0.16.

```
spacings(
  ( minEndOfNotchSpacing "Metal1"
    'notchWidth 0.15
    'notchLength 0.08
    'notchSpacing 0.16
    0.14
  )
) ;spacings
```

Metal1



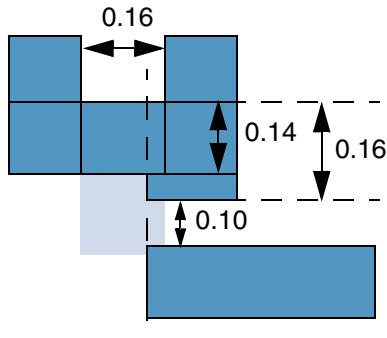
a) FAIL. Notch width is 0.14 (<0.15), notch length is 0.08, and notch spacing is 0.16, but the spacing between the two shapes is only 0.13 (<0.14).



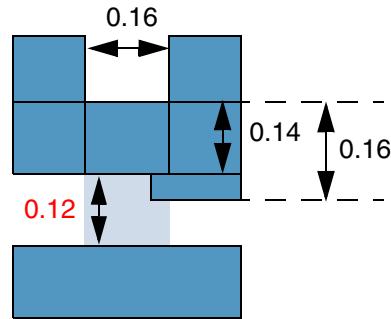
b) The constraint does not apply because there is no overlap with the notch.

## Virtuoso Technology Data Constraint 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

```

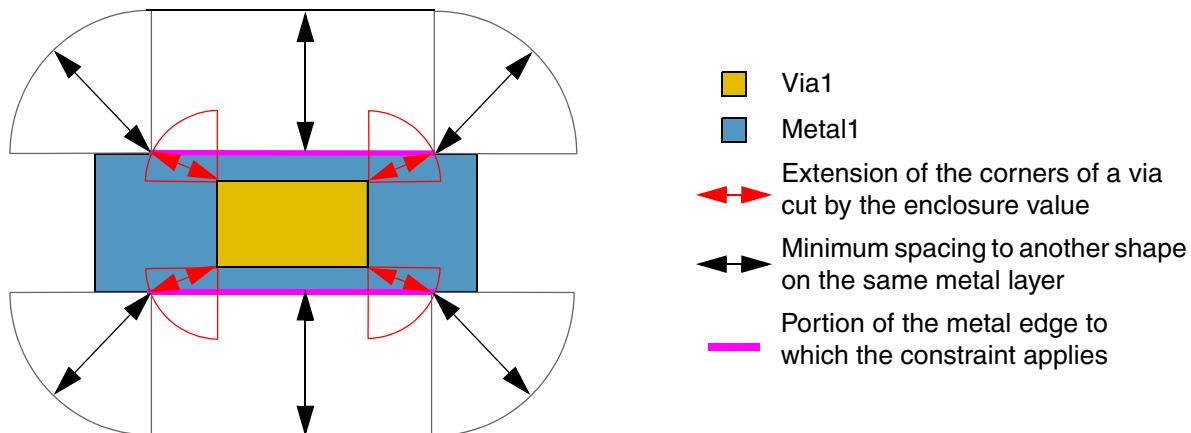
spacingTables(
  ( minExtensionSpacing tx_layer
    (( "extension" nil nil )
     ['cutClass {f_width | (f_width f_length) | t_name} ['longEdgeOnly]]
     ['layer tx_cutLayer]
     [f_default]
    )
    (g_table)
  )
) ;spacingTables

```

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 Constraint 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: $(f\_extension \ f\_spacing)$ 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 <u><a href="#">cutClasses</a></u> constraint).
	<ul style="list-style-type: none"><li>■ <i>f_width</i>: Width</li><li>■ <i>f_length</i>: Length</li><li>■ <i>t_name</i>: Name of the cut class</li></ul>
'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 Constraint Reference

### Spacing Constraints (One Layer)

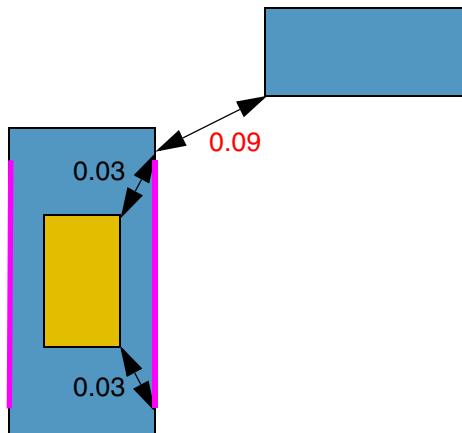
---

#### Example

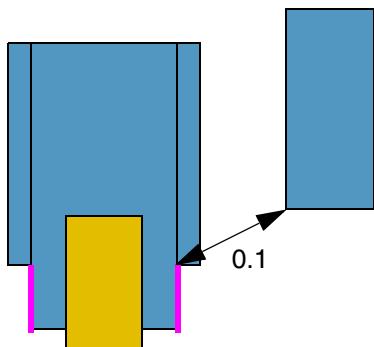
The spacing between a Metal1 edge that has an extension less than 0.03 past a cut shape of type VB (0.04x0.06) on layer Via1 to another Metal1 shape must be at least 0.1.

```
spacingTables(
  ( minExtensionSpacing "Metal1"
    (( "extension" nil nil )
      'cutClass "VB" 'longEdgeOnly
      'layer "Via1"
    )
    (0.03 0.1)
  )
) ;spacingTables
```

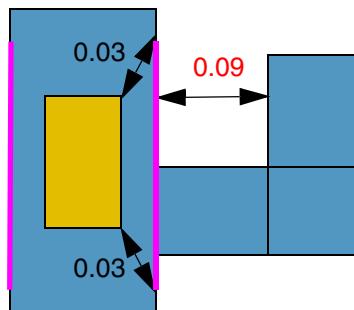
 Via1  
 Metal1



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 actual 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.

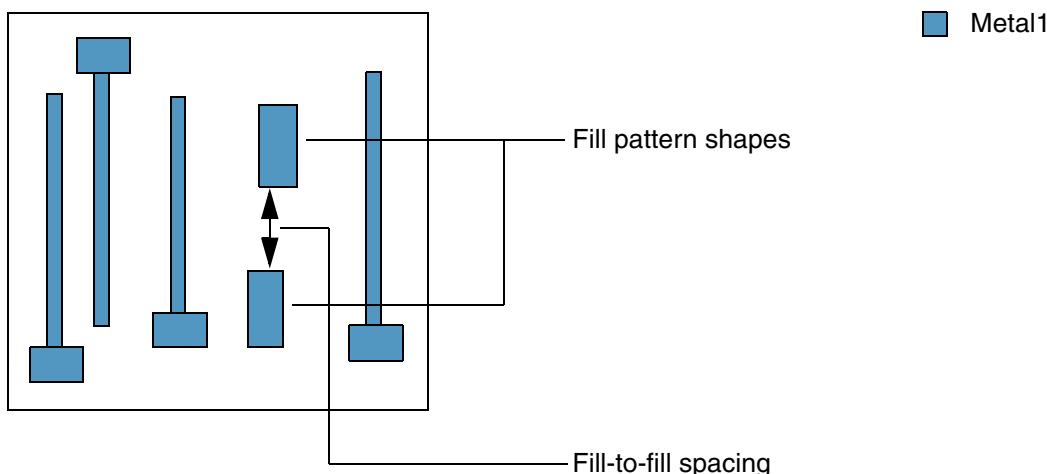


c) FAIL. The spacing between the portion of the metal edge to which the constraint applies (in pink) and another edge of the contiguous same-metal shape must be 0.1, but the actual spacing is only 0.09 (<0.1).

## 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 Constraint Reference

### Spacing Constraints (One Layer)

---

#### Example

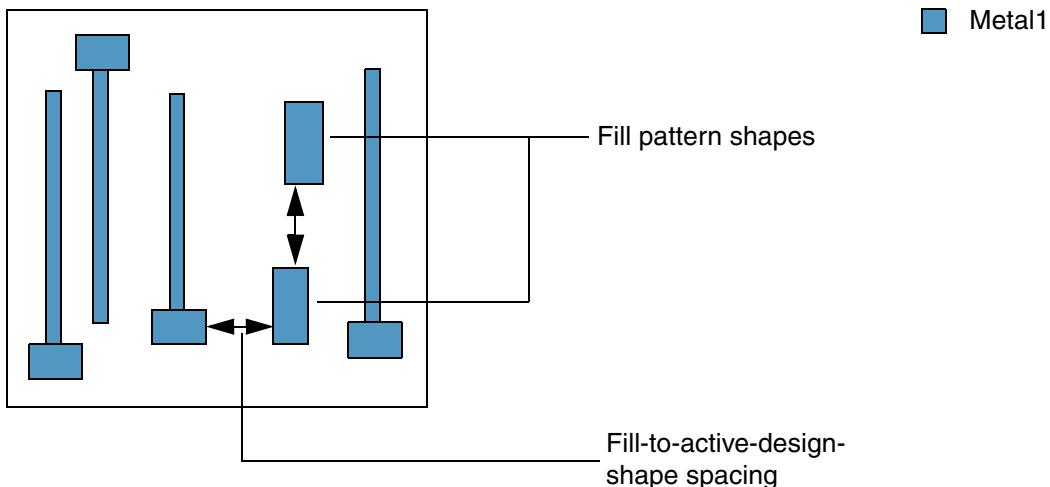
The minimum spacing between fill shapes on Metal1 must be at least 0.2.

```
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 Constraint 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 (ICADV12.3 Only)**

```
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
```

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.

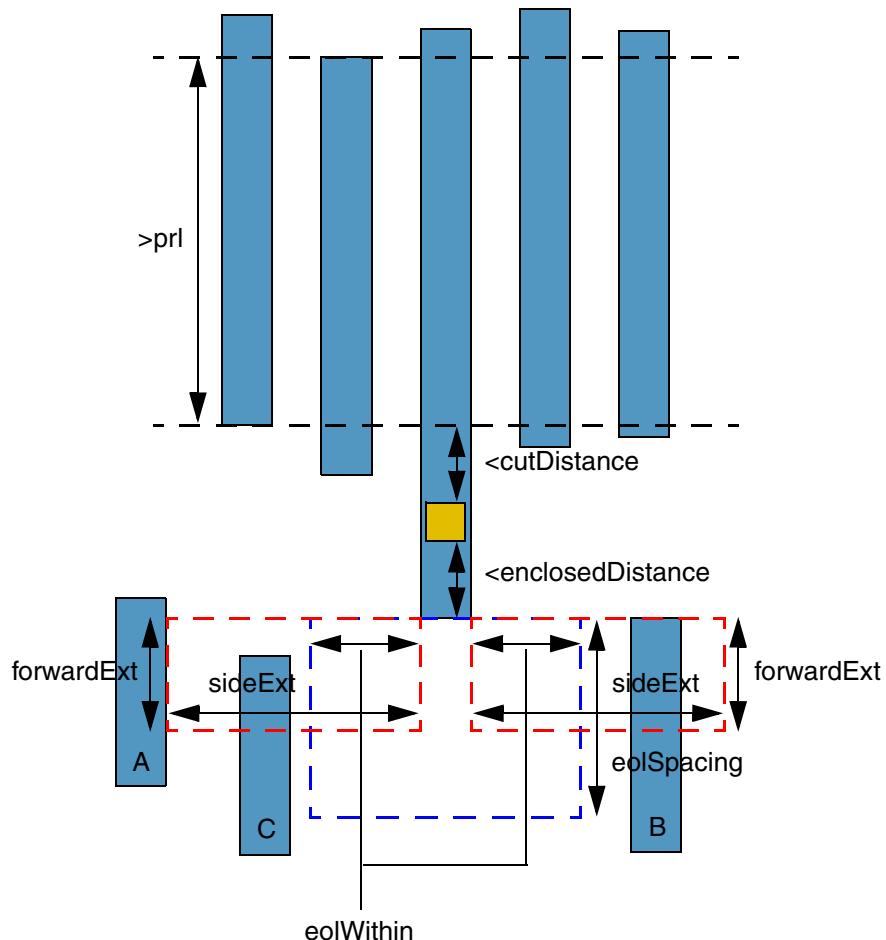
## Virtuoso Technology Data Constraint Reference

### 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 Constraint 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>pr1</i> .
'pr1 <i>f_pr1</i>	The constraint applies only if the five wires have common parallel run length greater than this value.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

```
'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
```

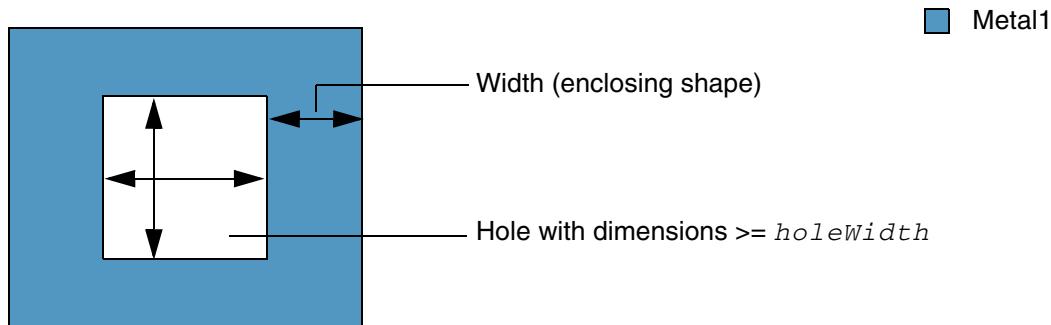
## minHoleWidth

```
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

<i>tx_layer</i>	The layer on which the constraint is applied. Type: String (layer and purpose names) or Integer (layer number)
<i>f_holeWidth</i>	The width of the hole in both X and Y directions must be greater than or equal to this value.

"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
```

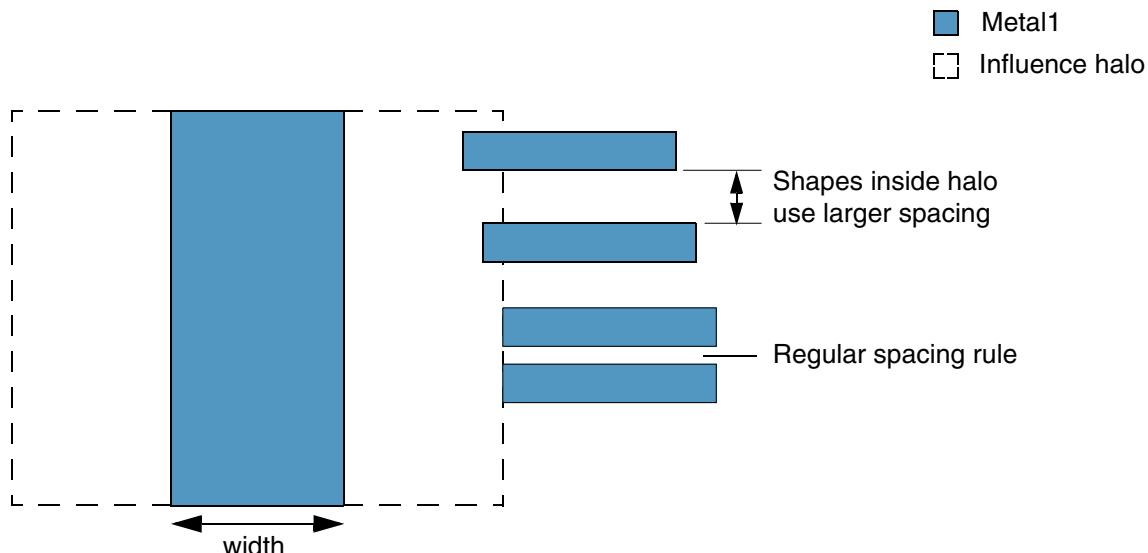
## minInfluenceSpacing

```
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 very 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 Constraint 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:  $(f\_width \ f\_spacing)$ 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:  $(f\_width \ f\_distance) \ f\_spacing$ where, <ul style="list-style-type: none"><li>■ <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.</li><li>■ <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.</li></ul> 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 Constraint 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**

'jointWidth <i>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.
'spanLength <i>f_spanLength</i>	The constraint applies only if the span length of the joint is greater than this value.
'sameMask	(Advanced Nodes Only) The constraint applies only between shapes on the same mask.  Type: Boolean
'length <i>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 Constraint Reference

### Spacing Constraints (One Layer)

'minLength *f\_minLength*

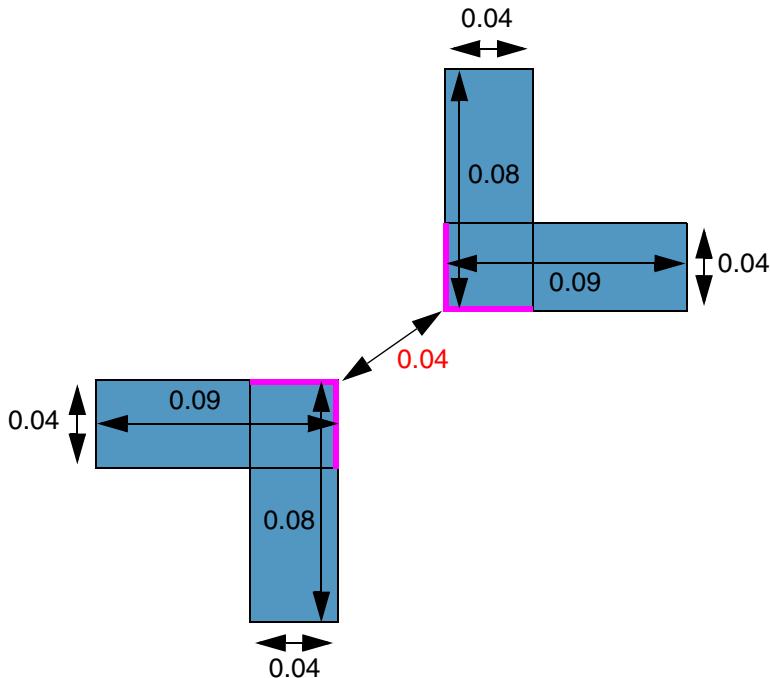
The constraint applies only if the length of both sides of a joint (that is not an end-of-line edge) is greater than or equal to this value.

#### 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



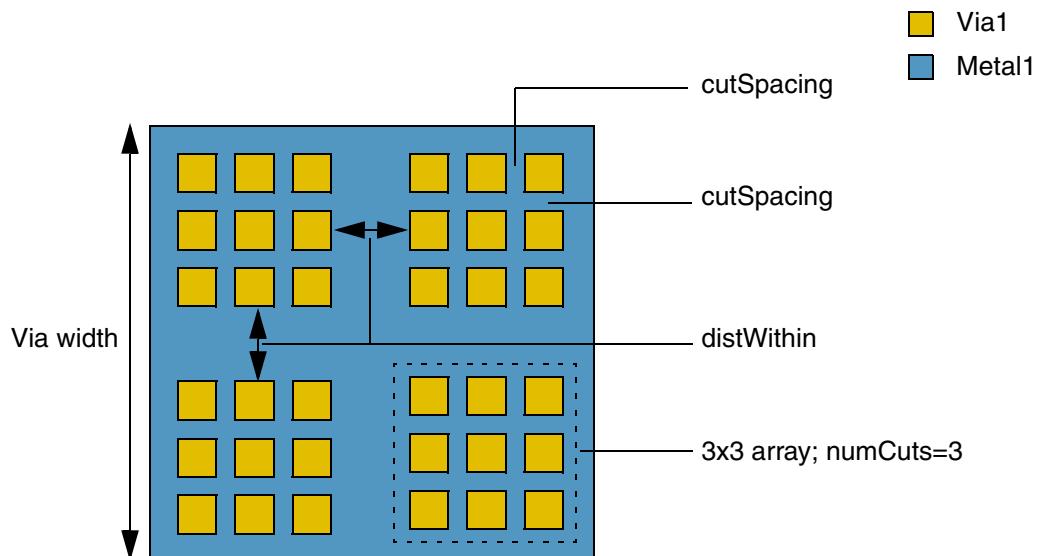
FAIL. The joint width is 0.04 (<0.07), the span lengths are 0.08 and 0.09 (both greater than 0.06), and the lengths of the edges that form the corners are both less than 0.1. However, the spacing between the corners is only 0.04 (<0.05).

## 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 width greater than or equal to *arrayWidth*, which is measured in the direction perpendicular to *distWithin*.



### Values

<code>tx_cutLayer</code>	The layer on which the constraint is applied.  Type: String (layer and purpose names) or Integer (layer number)
<code>f_cutSpacing</code>	The spacing between via cuts must be greater than or equal to this value.

## 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 width 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 Constraint Reference

### Spacing Constraints (One Layer)

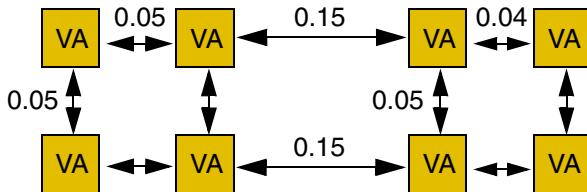
---

#### Example

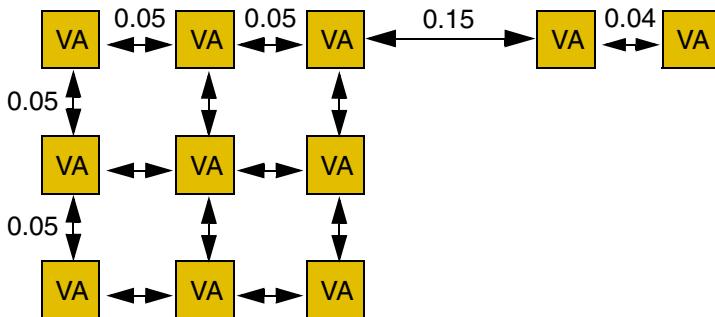
The spacing between VA via cuts in a 3x3 array and in a neighboring array at a distance less than 0.2 from the large array with width greater than or equal to 0.06 must be at least 0.05.

```
spacings(
  ( minLargeNeighborViaArrayCutSpacing "Via1"
    'numCuts 4
    'cutClass "VA"
    'distanceWithin 0.2 'arrayWidth 0.06
    0.05
  )
) ;spacings
```

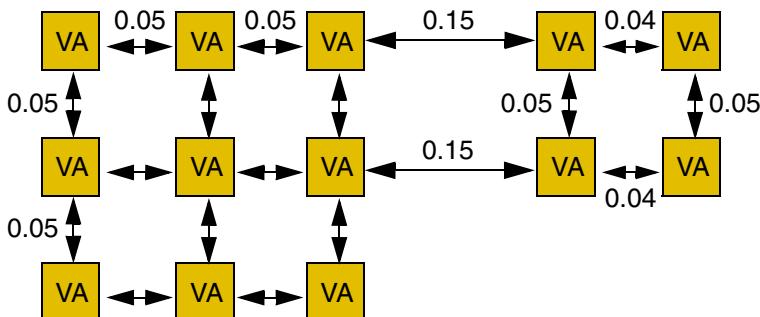
Via1



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).

## minNotchSpacing

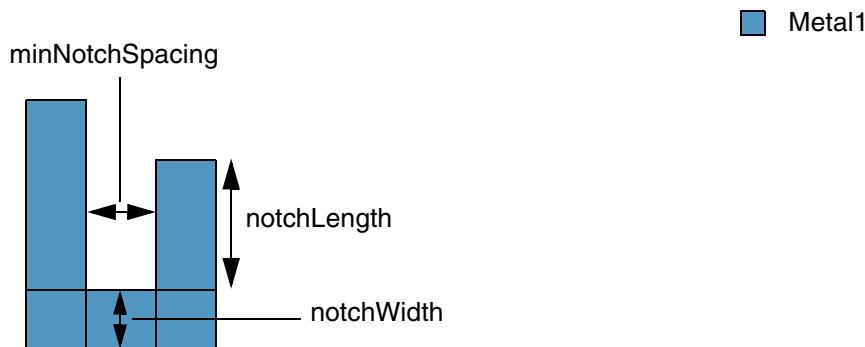
```

spacings(
  ( minNotchSpacing tx_layer
    '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

*tx\_layer*      The layer on which the constraint is applied.

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

*f\_spacing*      The minimum notch spacing.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

#### Parameters

'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*

(Advanced Nodes Only) The constraint applies only if the width of the notch is less than this value.

'excludeSpacing (*g\_range*)

(ICADV12.3 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

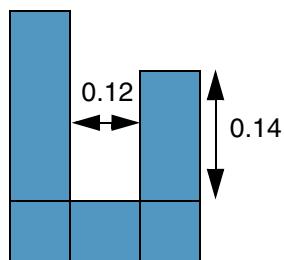
- [Example 1: minNotchSpacing with notchLength](#)
- [Example 2: minNotchSpacing with notchLength and notchWidth](#)
- [Example 3: minNotchSpacing with notchLength and concaveEndsWidth](#)
- [Example 4: minNotchSpacing with notchLength, within, and maxSpanLength](#)
- [Example 5: minNotchSpacing with notchWidth and excludeSpacing](#)

### ***Example 1: minNotchSpacing with notchLength***

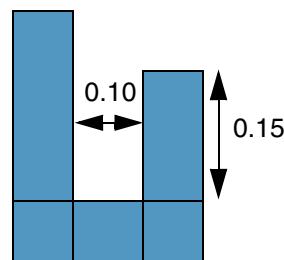
The notch spacing must be at least 0.12 if the shorter side of the notch is less than 0.15 long.

```
spacings(
  ( minNotchSpacing "Metal1"
    'notchLength 0.15
    0.12
  )
) ;spacings
```

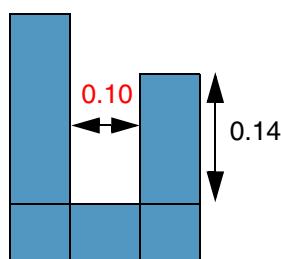
█ Metal1



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 Constraint Reference

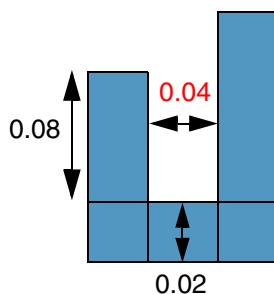
### Spacing Constraints (One Layer)

#### **Example 2: minNotchSpacing with notchLength and notchWidth**

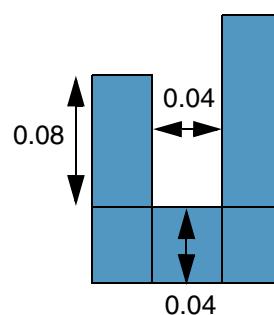
The notch spacing must be at least 0.05 if the shorter side of the notch is less than 0.09 long and the notch width is less than 0.03.

```
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).

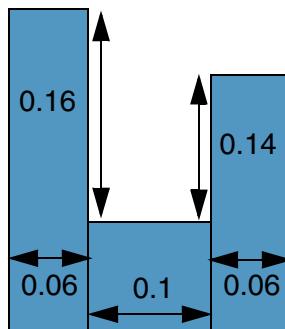
**Example 3: minNotchSpacing with notchLength and concaveEndsWidth**

The notch spacing must be at least 0.12 if the following conditions are met:

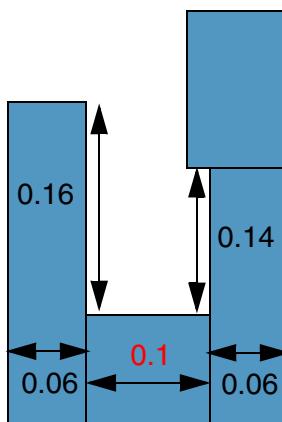
- Both sides of the notch are less than or equal to 0.08 wide.
- The shorter side is less than 0.15 long and lies between two concave corners, and the longer side is greater than or equal to 0.1.

```
spacings(
  ( minNotchSpacing "Metall1"
    'notchLength 0.15
    'concaveEndsWidth 0.08
    0.12
  )
) ;spacings
```

■ Metal1



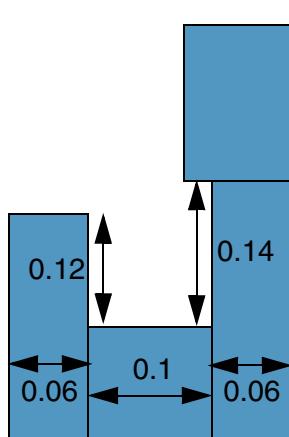
a) The constraint does not apply because the shorter side does not lie between two concave corners. A violation would occur if  
'concaveEndsWidth was not specified.



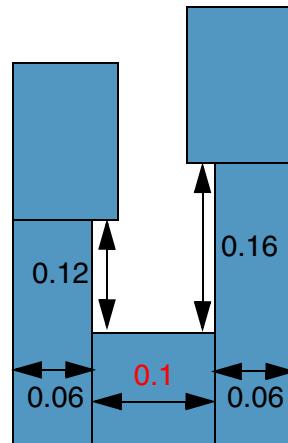
b) FAIL. Both sides of the notch are less than 0.08 wide; the shorter side is 0.14 (<0.15) and lies between two concave corners, and the longer side is 0.16 (>0.15). However, the notch spacing is 0.1 (<0.12).

## Virtuoso Technology Data Constraint 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).

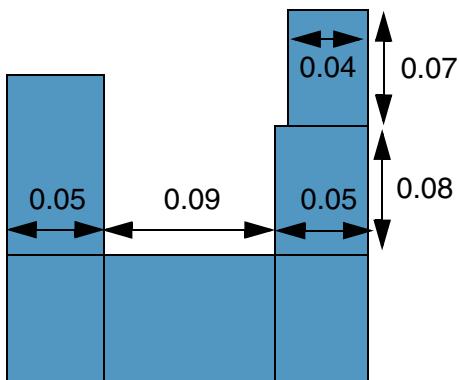
**Example 4: minNotchSpacing with notchLength, within, and maxSpanLength**

The notch spacing must be at least 0.1 if the following conditions are met:

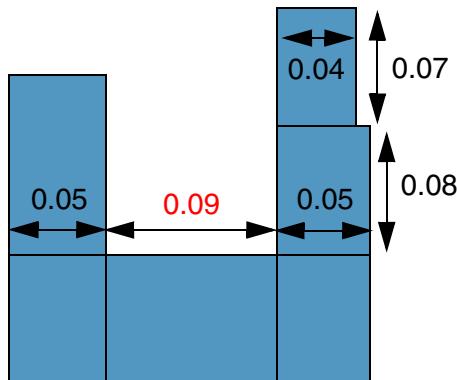
- The span length of at least a portion of the side that is within 0.2 of the notch is less than 0.05.
- The length of this portion is greater than or equal to 0.06.

```
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).

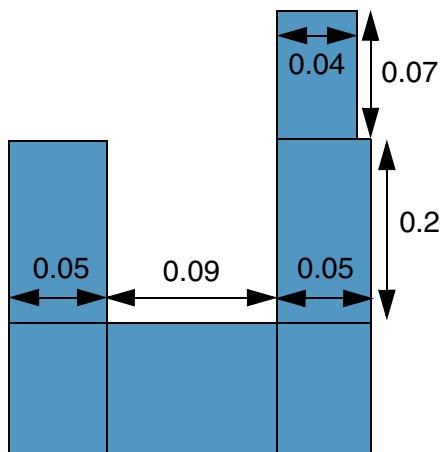


b) FAIL. The top portion of the right side of the notch is within 0.2 of the notch. Its span length is 0.04 (<0.05) and its length is 0.07 (>0.06), but the notch spacing is 0.09 (<0.1).

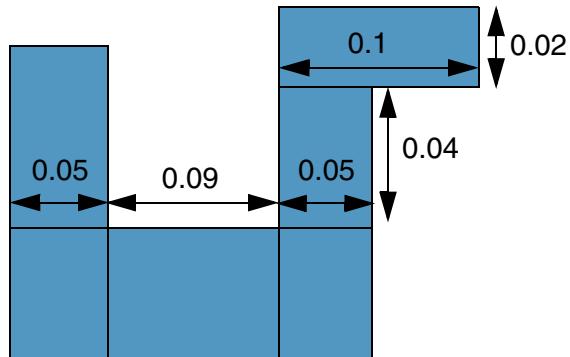
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

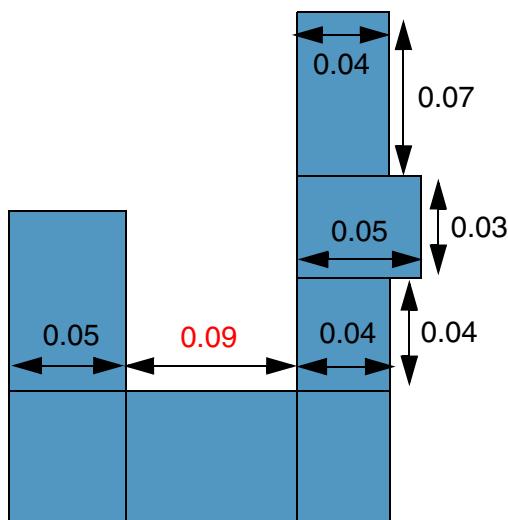
---



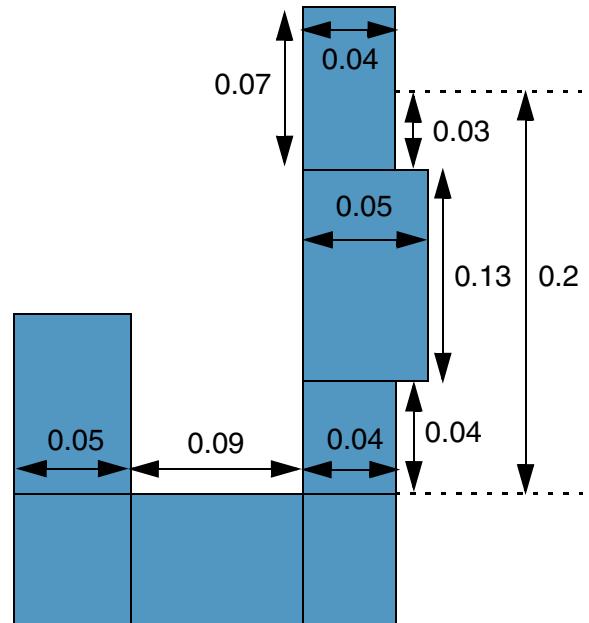
c) The constraint does not apply because the span length of the bottom portion of the right side of the notch (the portion that is within 0.2 of the notch) is equal to 0.05 (must be less than 0.05).



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 Constraint Reference

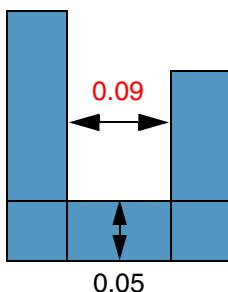
### Spacing Constraints (One Layer)

#### **Example 5: minNotchSpacing with notchWidth and excludeSpacing**

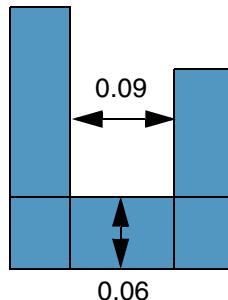
The notch spacing must be at least 0.1 if the notch width is less than or equal to 0.05. The constraint does not apply if the notch spacing is greater than or equal to 0.06 and less than 0.08.

```
spacings(
  ( minNotchSpacing "Metall1"
    'notchWidth 0.05
    'excludeSpacing "[0.06 0.08)"
    0.1
  )
) ;spacings
```

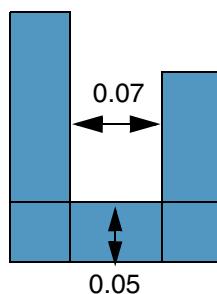
 Metal1



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 exactly equal to this value.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

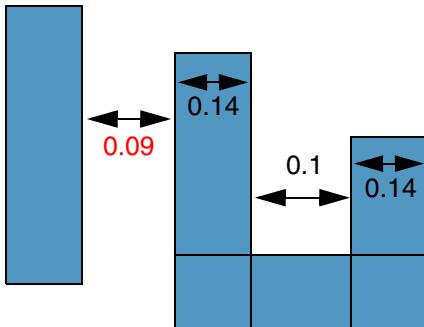
---

#### Example

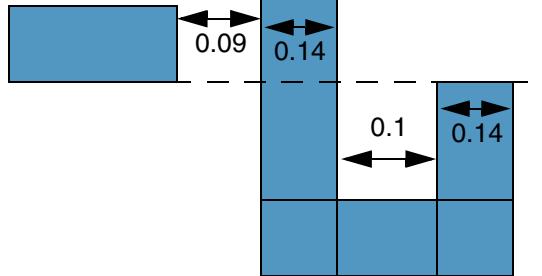
The spacing between a notch and a neighboring shape must be at least 0.1 if the span of the legs of the notch is less than 0.15 and the spacing between the legs of the notch is less than 0.12. Notches with length exactly equal to 0.13 are exempt from the rule.

```
spacings(
  ( minNotchSpanSpacing "Metal1"
    'notchSpan 0.15
    'notchSpace 0.12
    'exceptNotchLength 0.13
    0.1
  )
) ; spacings
```

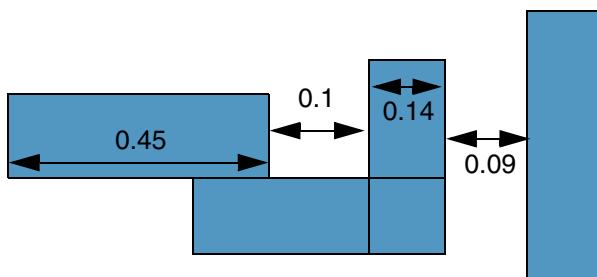
Metal1



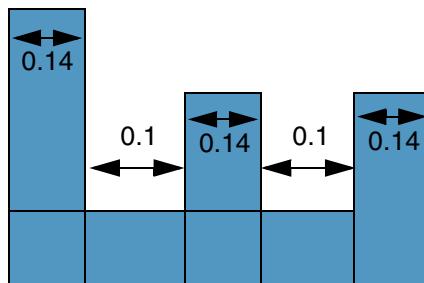
a) FAIL. The span of the legs of the notch is 0.14 (<0.15) and the distance between the legs of the notch is 0.1 (<0.12). However, the distance of the neighboring shape from the notch is only 0.09 (<0.1).



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.



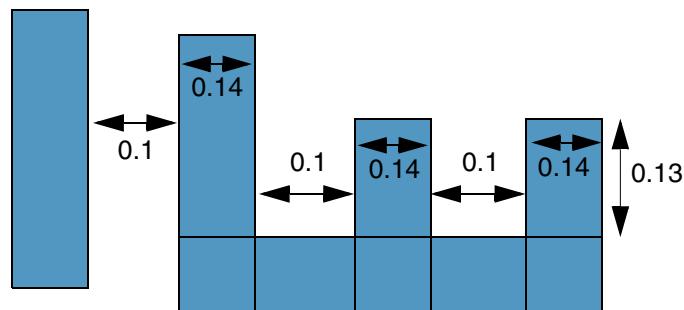
c) The constraint does not apply because the span of the left leg of the notch is 0.45 (>0.15).



d) FAIL. Multiple consecutive notches cause a violation, irrespective of the distance of the neighboring shape from a notch.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)



- e) The constraint does not apply because the length of a leg of the notch is exactly equal to 0.13.

## 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.

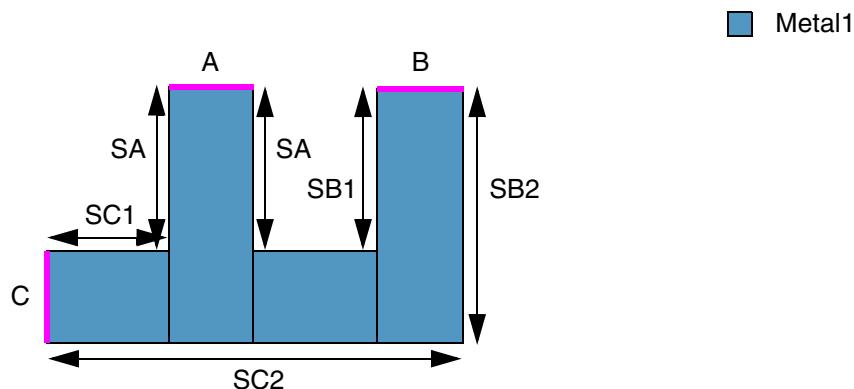
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

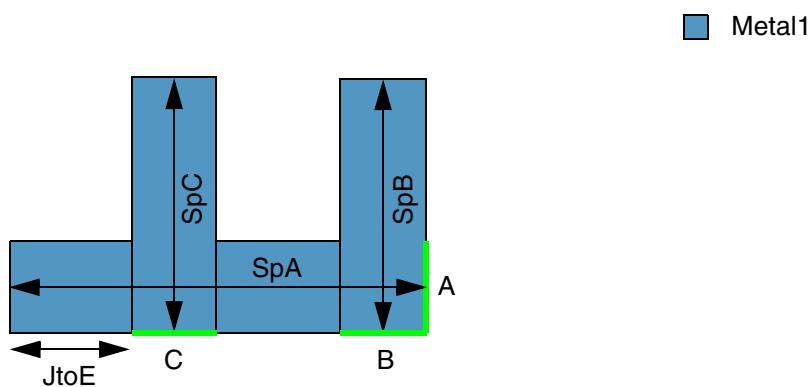
In an end-to-joint or joint-to-end scenario, both `endToJoint` and `jointToEnd` conditions must be fulfilled individually, and to fulfill a condition, either joint spacing must be greater than or equal to `jointSpacing` or end spacing must be greater than or equal to `endSpacing`.

The figure below illustrates end-of-line wires.



Edges labeled A, B, and C represent end-of-line edges and SA, SB<sub>n</sub>, and SC<sub>n</sub> 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.

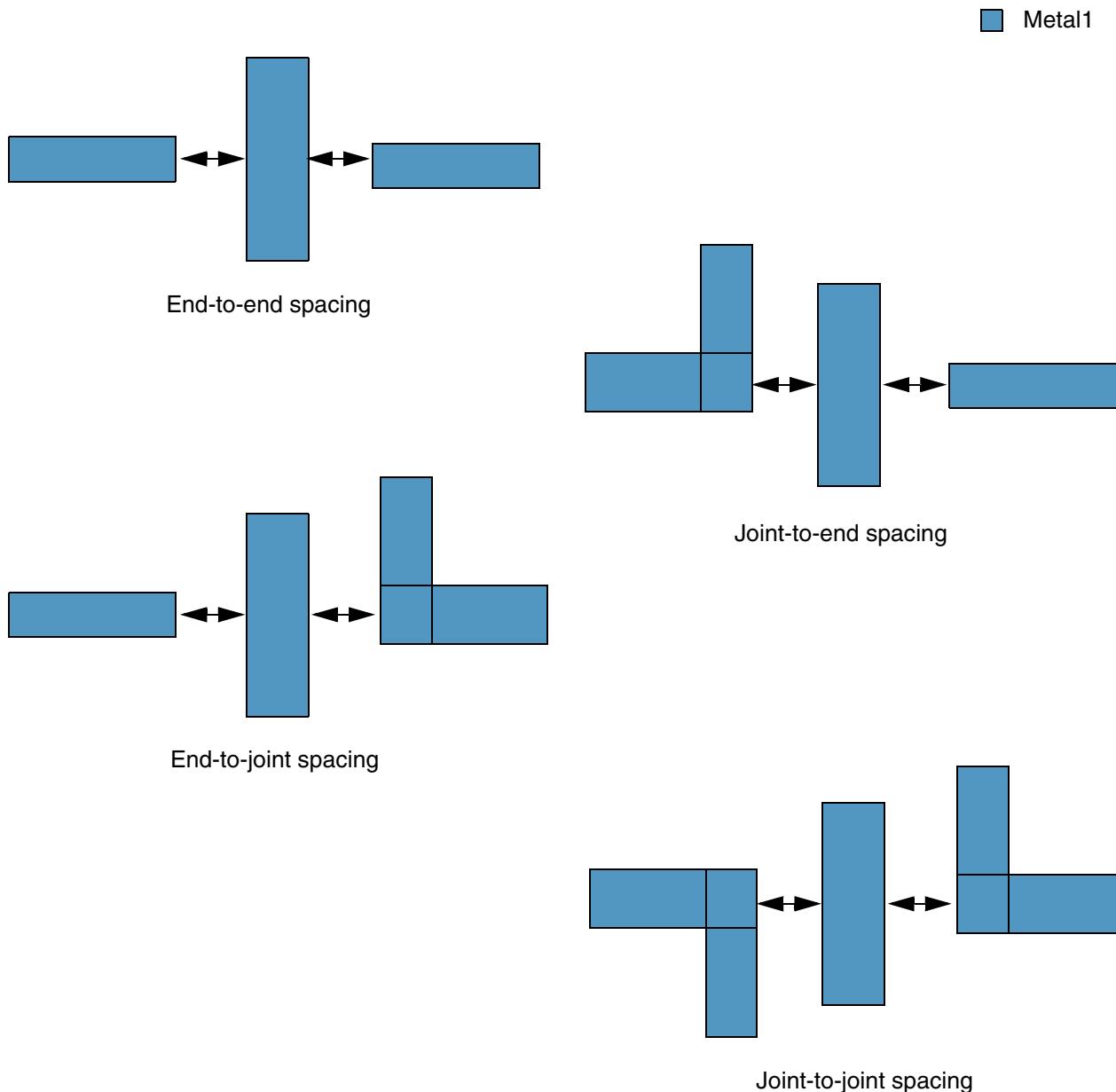


Edges labeled A, B, and C are joints if their width is less than `eolWidth` or `jointWidth` and the span lengths SpA, SpB, and SpC, respectively, are greater than `jointLength`. The constraint applies only if the distance JtoE, measured from the joint to the end of the edge that contains the joint, is 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 Constraint Reference

### Spacing Constraints (One Layer)

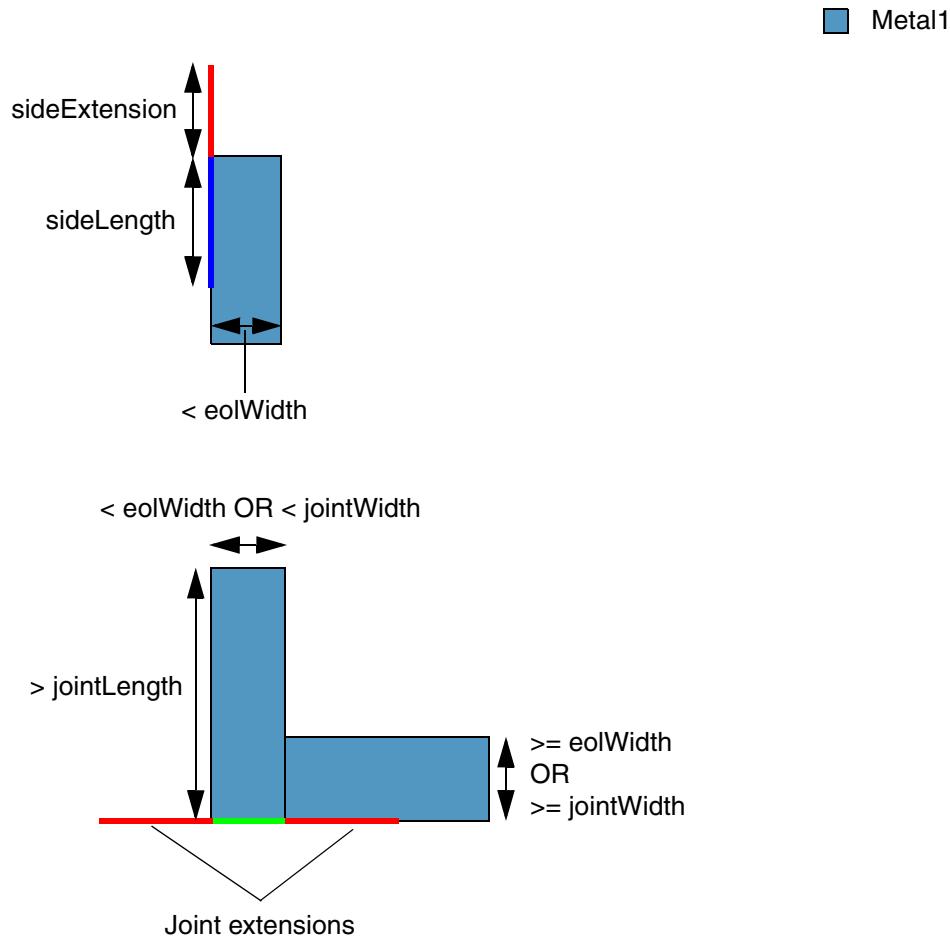
The figures below illustrate the four different types of spacing. The width of the middle wire must be less than *width*.



## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

The following figures illustrate side and joint extensions. The green edge depicts a joint.

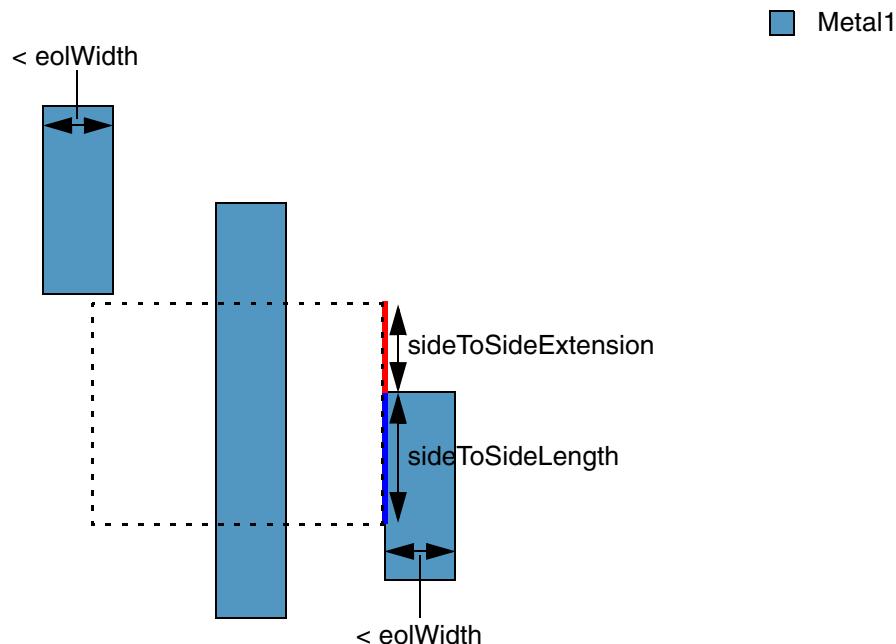


## Virtuoso Technology Data Constraint Reference

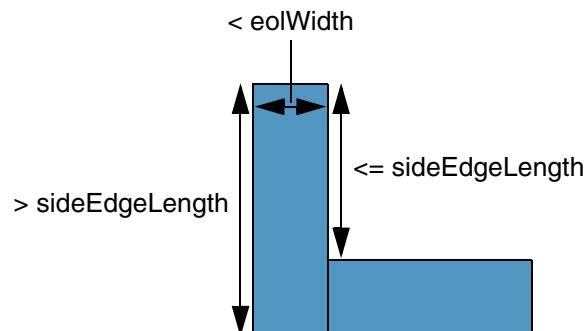
### Spacing Constraints (One Layer)

---

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.



`sideExtension` and `sideLength` are applied to one side. The constraint applies only if another side is present inside the search window depicted by the dotted region.



Only the left edge is greater than `sideEdgeLength` and qualifies as a side.

## Virtuoso Technology Data Constraint 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)

*endToEnd f\_endSpacing1 f\_endSpacing2*

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.

*jointToEnd f\_jointSpacing f\_endSpacing*

The joint spacing must be at least *jointSpacing* and the end spacing must be at least *endSpacing*.

Both *endToJoint* and *jointToEnd* spacing must be fulfilled individually.

*endToJoint f\_endSpacing f\_jointSpacing*

The end spacing must be at least *endSpacing* and the joint spacing must be at least *jointSpacing*.

Both *endToJoint* and *jointToEnd* spacing must be fulfilled individually.

*jointToJoint f\_jointSpacing1 f\_jointSpacing2*

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.

*sideToEnd f\_sideSpacing f\_endSpacing*

The spacing between a side that satisfies '*sideLength*' and the middle wire must be at least *sideSpacing* and between an end-of-line edge and the middle wire must be at least *endSpacing*.

*sideToJoint f\_sideSpacing f\_jointSpacing*

The spacing between a side that satisfies '*sideLength*' and the middle wire must be at least *sideSpacing* and between a joint and the middle wire must be at least *jointSpacing*.

## Virtuoso Technology Data Constraint 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` (ICADV12.3 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 Constraint 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 Constraint 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.

'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 Constraint Reference

### Spacing Constraints (One Layer)

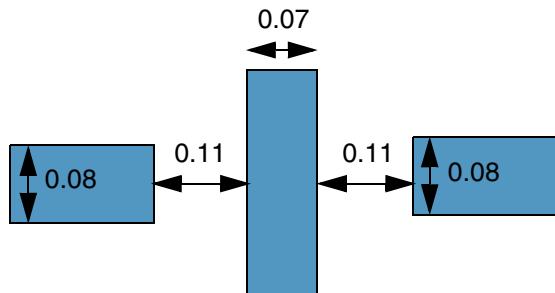
#### **Example 1: minOppSpanSpacing with width, eolWidth, jointLength, and exceptEdgeLength**

The constraint applies if the following conditions are satisfied:

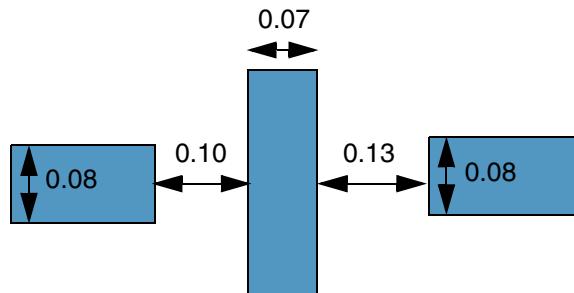
- Width of the middle wire is less than 0.08
- End-of-line width of an end wire is less than 0.1
- Span length of the joint is greater than 0.15

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
```



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.

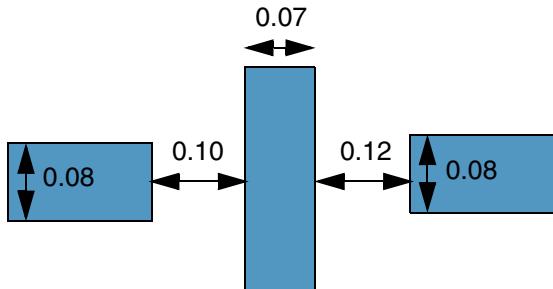


b) PASS. The spacing on the right is 0.13, which is equal to the larger of the two specified spacing values.

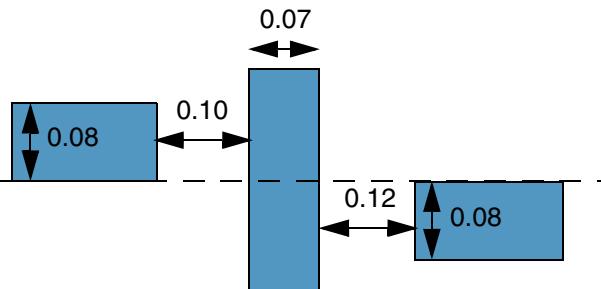
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

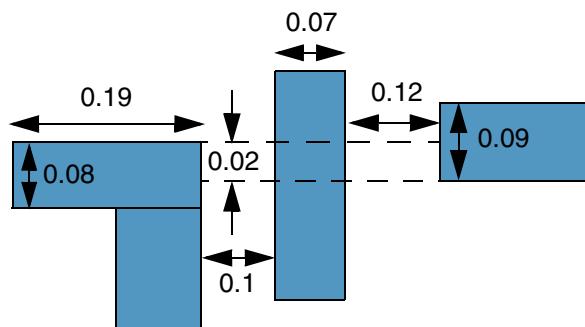
---



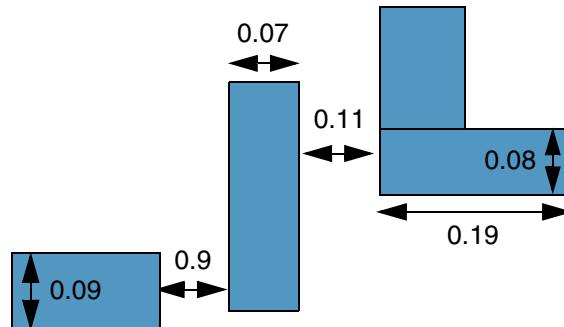
c) FAIL. The spacing on the left is 0.10, which is less than the smaller of the two specified spacing values.



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 Constraint Reference

### Spacing Constraints (One Layer)

---

#### **Example 2: minOppSpanSpacing with width, eolWidth, jointLength, sideLength, and prl**

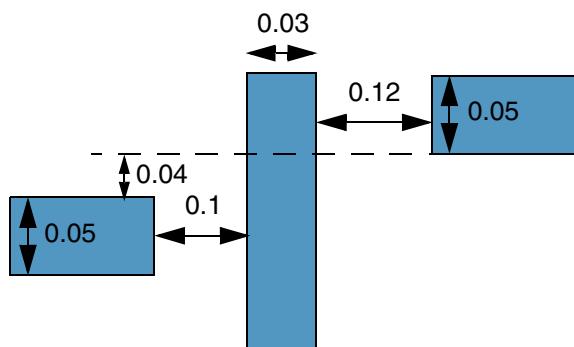
The constraint applies if the following conditions are satisfied:

- 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

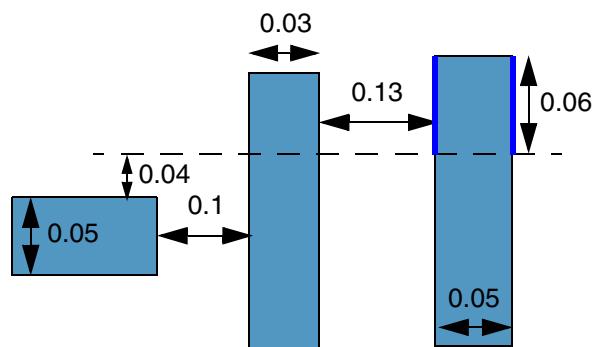
```

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

```



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 ( $<0.13$ ).



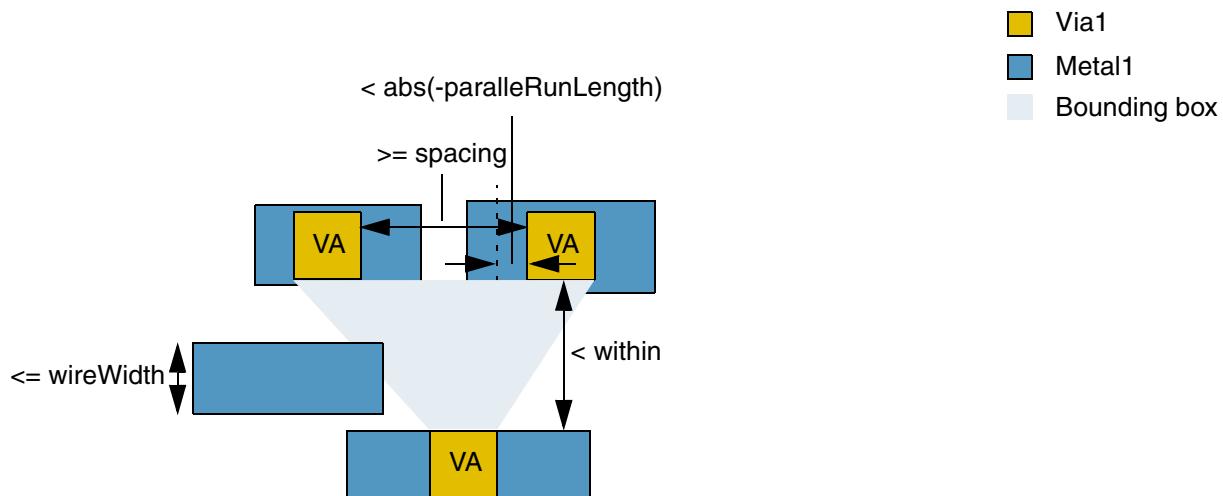
b) FAIL. The thick blue edges are sides of an end-of-line edge, but the distance of the side from the middle wire is 0.13 ( $<0.14$ ). Additionally, the distance of the end-of-line edge from the middle wire is 0.1 ( $<0.15$ ).

## minOrthogonalSpacing (ICADV12.3 Only)

```
orderedSpacings(
    ( minOrthogonalSpacing tx_cutLayer tx_metalLayer
        ['cutClass {f_width | (f_width f_length) | t_name}']
        'prl f_parallelRunLength 'within f_within
        'width f_wireWidth
        f_spacing
    )
)
; orderedSpacings
```

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 Constraint 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

'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

Otherwise, the constraint applies to via cuts of any cut class.

'prl *f\_parallelRunLength*

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 *within* from them, is greater than this value.

'within *f\_within*

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*

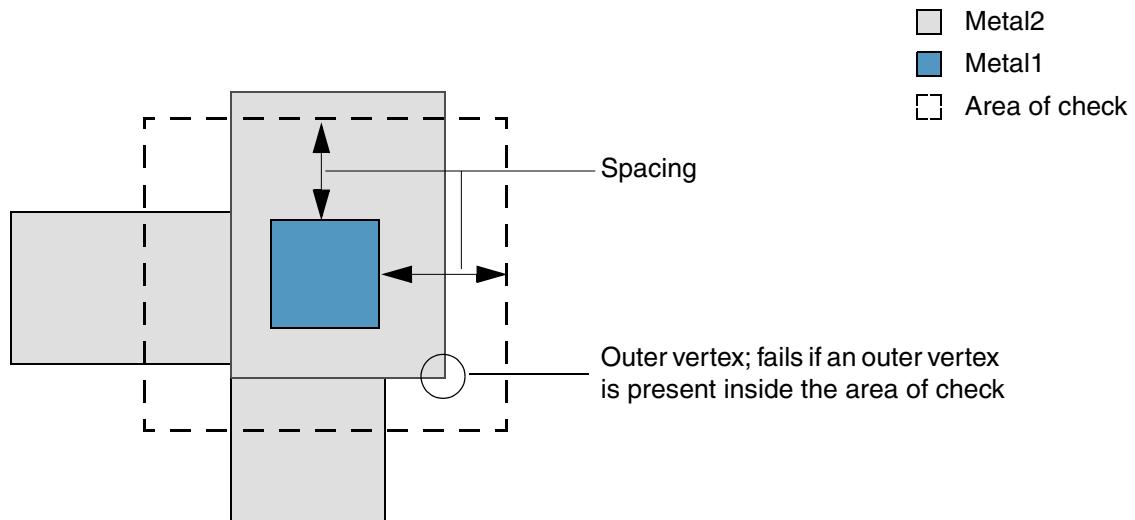
The constraint applies only if the width of the *metalLayer* wire is less than or equal to this value.

## **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 Constraint 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

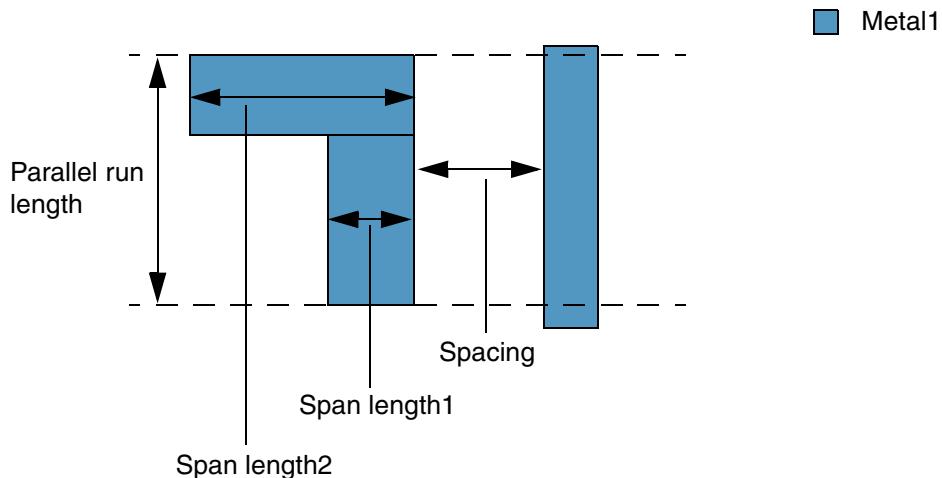
The spacing of a Metal2 shape from an outer vertex (convex corner) of a Metal1 shape must be at least 0.32.

```
orderedSpacings(  
    ( minOuterVertexSpacing "Metal2" "Metall1"  
        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.



## Values

*tx\_layer*      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 *table*.

*g\_table*      The format of a *table* row is as follows:

(*f\_index1 f\_index2*) *f\_value*

where,

- *f\_index1* is the span length of the first shape.
- *f\_index2* is the span length of the second shape.
- *f\_value* 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 (*index1*). Similarly, you need to locate the last column for which all span lengths of the second shape are greater than the index value (*index2*). 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 exactly equal to the number of rows. The span length and spacing values must increase from top to bottom and from left to right.

## Parameters

*f\_length*

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.

*f\_default*

The spacing value to be used when no table entry applies.

## **Virtuoso Technology Data Constraint 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:

<b>Spacing</b>	<b>index2 &gt;</b>	<b>0.00</b>	<b>0.10</b>	<b>0.20</b>
<b>index1 &gt;</b>				
<b>0.00</b>		0.10	0.15	0.20
<b>0.10</b>		0.15	0.17	0.23
<b>0.20</b>		0.20	0.23	0.25

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

```

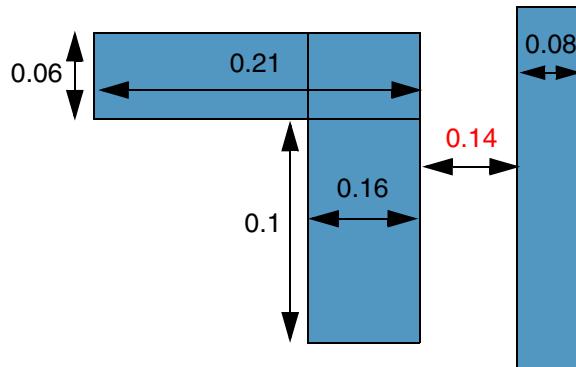
spacingTables(
  ( minParallelSpanSpacing "Metal1"
    (( "span" nil nil "span" nil nil )
     'length 0.15
    )
    (
      (0.00 0.00) 0.10
      (0.00 0.10) 0.15
      (0.00 0.20) 0.20

      (0.10 0.00) 0.15
      (0.10 0.10) 0.17
      (0.10 0.20) 0.23

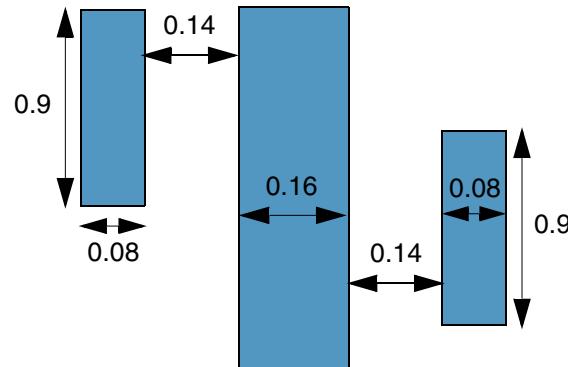
      (0.20 0.00) 0.20
      (0.20 0.10) 0.23
      (0.20 0.20) 0.25
    )
  )
)
; spacingTables

```

  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 actual spacing is only 0.14.

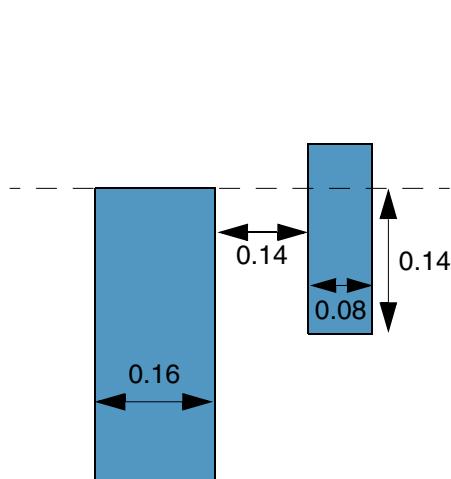


b) The constraint does not apply because the parallel run length with the shapes to the left and right is only 0.9 ( $< 0.15$ ). Parallel run lengths of shapes on opposite sides do not add up.

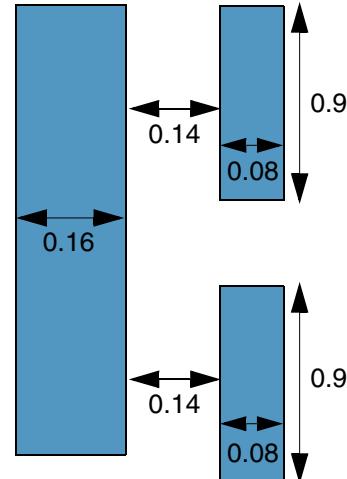
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

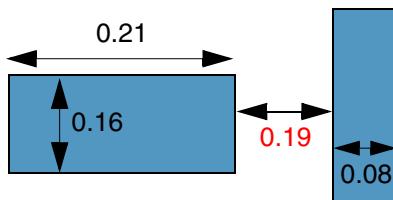
---



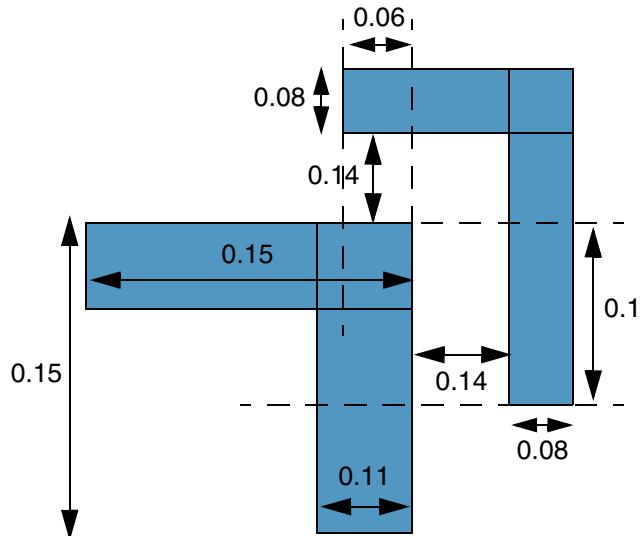
c) The constraint does not apply because the parallel run length between the two shapes is only 0.14 (<0.15).



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.



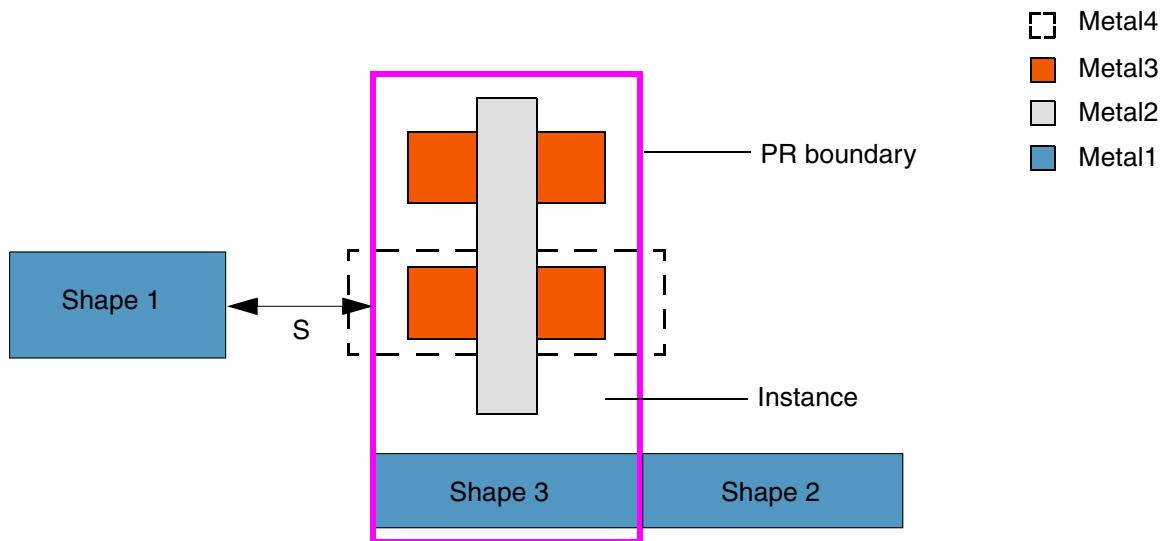
f) The constraint does not apply because the top and side parallel run lengths are both less than 0.15.

## **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. Note that 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.



## 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.

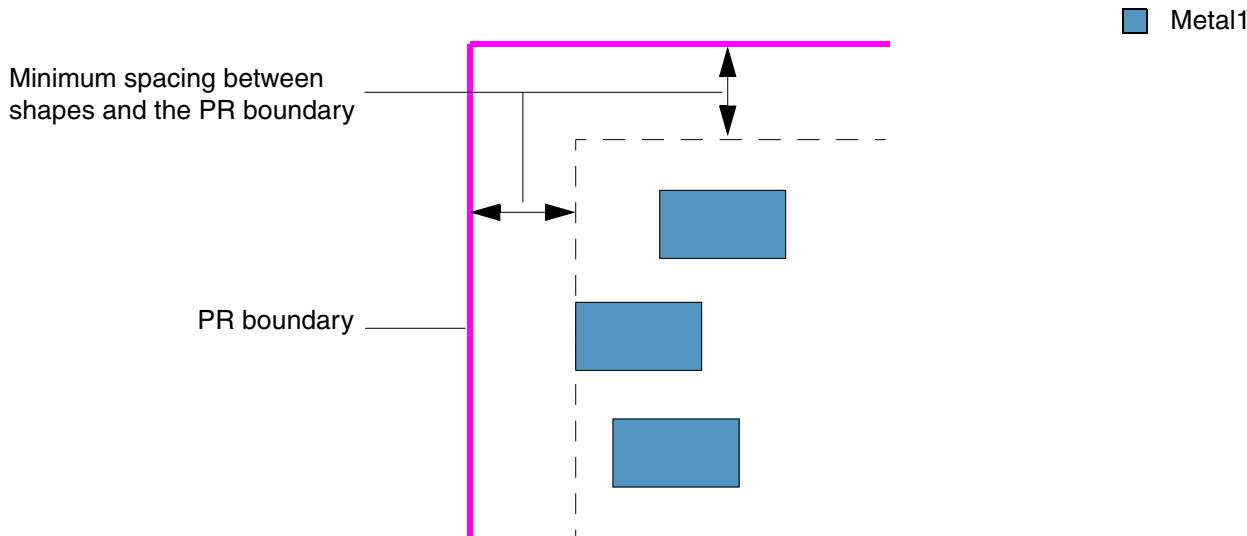
```
spacings(
  ( minPRBoundaryExteriorHalo "Metal1"
    1.0
  )
) ;spacings
```

## **minPRBoundaryInteriorHalo**

```
spacings(  
  ( minPRBoundaryInteriorHalo tx_layer  
    ['horizontal | 'vertical]  
    ['stepSize f_stepSize]  
    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 Constraint Reference

### Spacing Constraints (One Layer)

---

#### Parameters

'horizontal | 'vertical

(Advanced Nodes Only) The direction in which spacing is measured. If direction is not specified, spacing is measured in any direction.

Type: Boolean

'stepSize *f\_stepSize*

(Advanced Nodes Only) The allowed spacing ( $S_{legal}$ ) is restricted to the following values:

$$S_{legal} = S_{min} + n * stepSize$$

where,  $S_{min}$  is the minimum spacing requirement.

'coincidentAllowed The shape can coincide with the PR boundary. Otherwise, the spacing requirement must be met.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

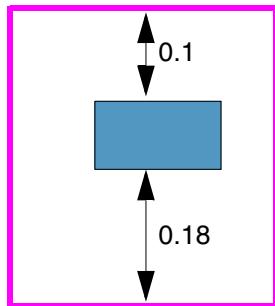
### Spacing Constraints (One Layer)

#### Example

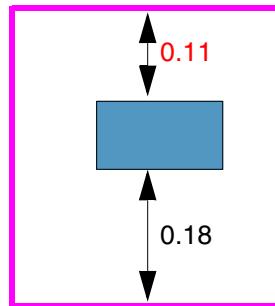
The vertical spacing between a Metal1 shape and the enclosing PR boundary must be at least 0.1 or equal to the sum of the constraint value (0.1) and an integer multiple of 0.04.

```
spacings(
  ( minPRBoundaryInteriorHalo "Metal1"
    'vertical
    'stepSize 0.04
    0.1
  )
) ;spacings
```

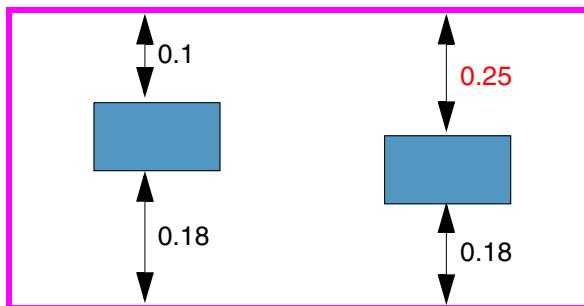
 Metal1  
 PR boundary



a) PASS. The upper vertical spacing is exactly equal to 0.1 and the bottom vertical spacing of 0.18 is equal to 0.1 plus an integer multiple of 0.04 ( $0.1+2\times0.04=0.18$ ).



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.

## **minPrlTwoSidesSpacing (ICADV12.3 Only)**

```
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
```

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**

*tx\_cutLayer*      The layer on which the constraint is applied.

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

*f\_prlSpacing f\_nonPrlSpacing*

The spacing between two cut shapes must be greater than or equal to *prlSpacing* or *nonPrlSpacing*.

### **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 Constraint 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 Constraint Reference

### Spacing Constraints (One Layer)

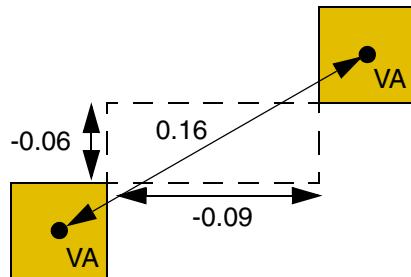
#### Example

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

*tx\_layer* The layer on which the constraint is applied.

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

`short f_shortJogSpacing`

If the parallel run length of the protrusion in either wire with the neighboring wire is less than or equal to  $protrusionLength$ , spacing between wires must be greater than or equal to this value.

`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)`

(Advanced Nodes Only) 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.

'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 Constraint Reference

### Spacing Constraints (One Layer)

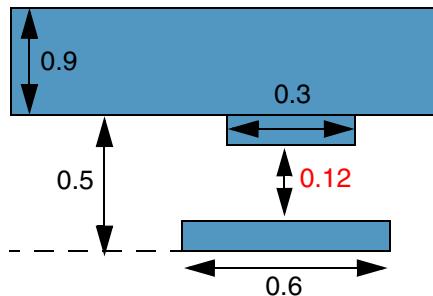
---

```

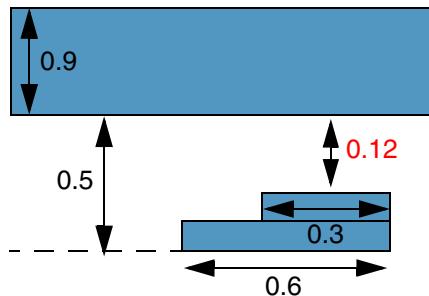
spacings(
  ( minProtrusionSpacing "Metall1"
    'width 0.4 'length 0.2 'within 0.5
    'protrusionLength 0.2
    (
      short 0.08
      long 0.13
      between 0.7
    )
  )
  ( minProtrusionSpacing "Metall1"
    'width 0.8 'length 0.3 'within 0.5
    'protrusionLength 0.2
    (
      short 0.08
      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, actual spacing is 0.12 ( $<0.15$ ).

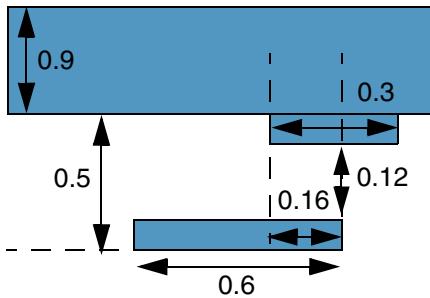


b) FAIL. All triggering conditions are met. The parallel run length of the protrusion with the neighboring wire is 0.3 ( $>0.2$ ); therefore, the spacing required is at least 0.15. However, actual spacing is only 0.12 ( $<0.15$ ).

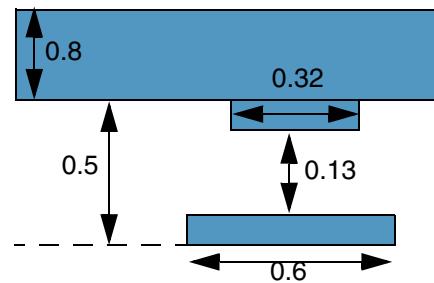
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

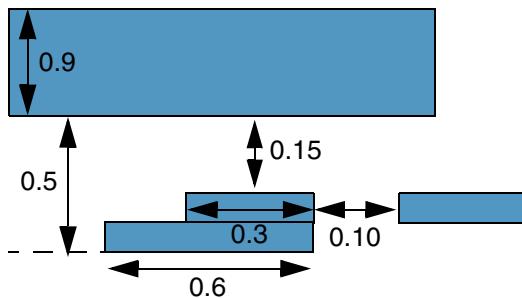
---



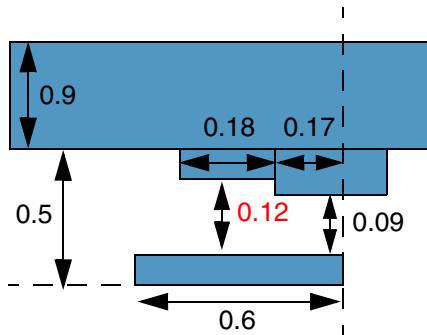
c) PASS. All triggering conditions are met. The parallel run length of the protrusion with the neighboring wire is 0.16 (<0.2); therefore, the spacing required is at least 0.08, which is met.



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.

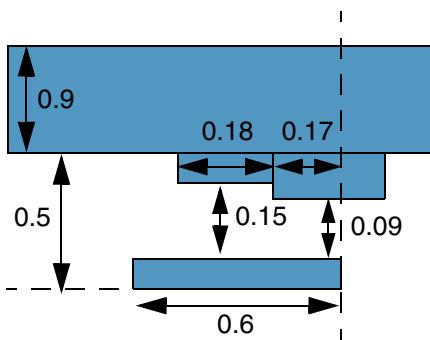


f) FAIL. All triggering conditions are met. The combined parallel run length of the protrusion with the neighboring wire is 0.35 (=0.18+0.17); therefore, the spacing required is at least 0.15, which is not met.

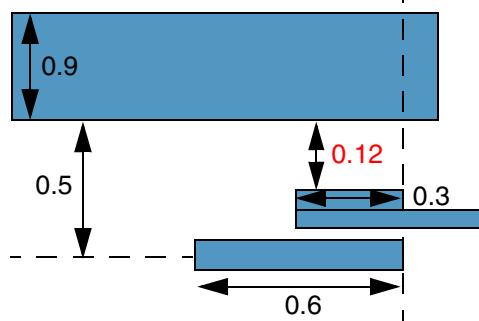
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

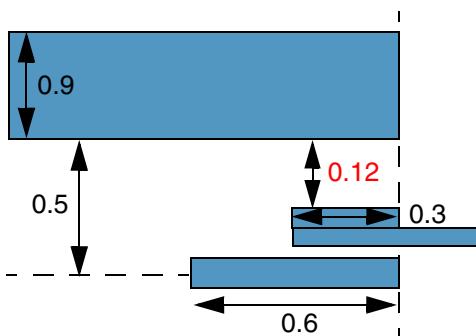
---



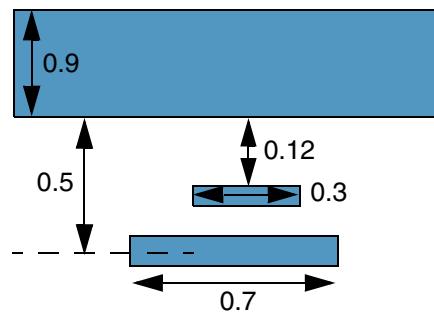
g) PASS. All triggering conditions are met. The combined parallel run length of the protrusion with the neighboring wire is 0.35 (=0.18+0.17); therefore, the spacing required is at least 0.15, which is met. The two 0.18 and 0.17 wide segments are both less than 0.2. This requires a spacing of at least 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.

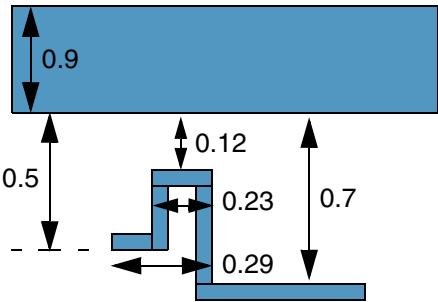


j) FAIL. The neighboring wire is 0.3 long and requires a spacing of at least 0.15 (it need not necessarily have a protrusion).

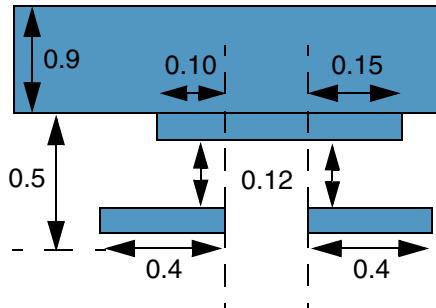
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

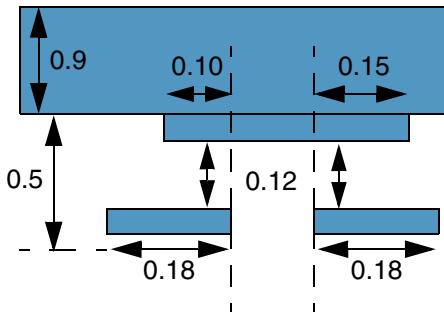
---



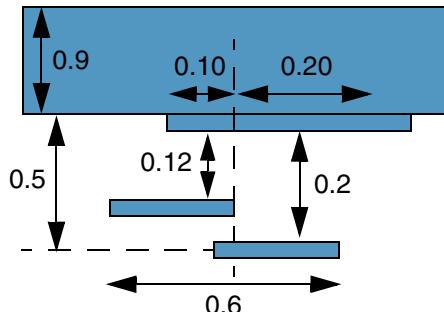
k) The constraint does not apply because the parallel run length of the portion that is within 0.5 of the wide wire is only 0.29 (<0.3).



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.

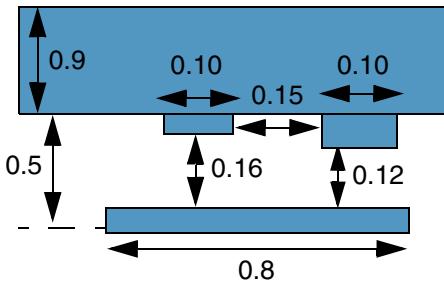


n) PASS. The combined parallel run length of the two neighboring wires is 0.6 (>0.3). The projected parallel run lengths of the two segments of the protrusion are 0.10 and 0.2 (<=0.2); therefore, a spacing of 0.08 is required and is met.

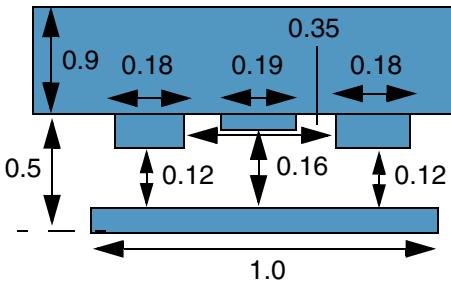
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

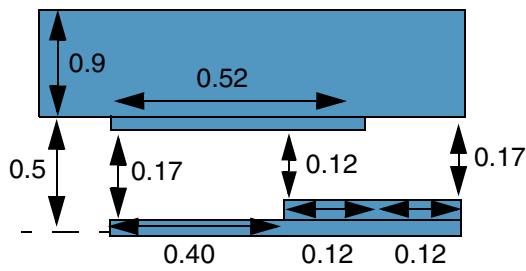
---



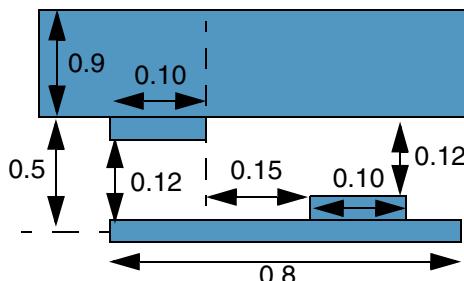
**o) PASS.** All triggering conditions and the spacing requirement between the wires is met. The spacing between the protrusions does not apply because the protrusion that has a spacing greater than 0.15 is only 0.10 wide (must be greater than 0.2 for the protrusion-to-protrusion spacing condition to 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.

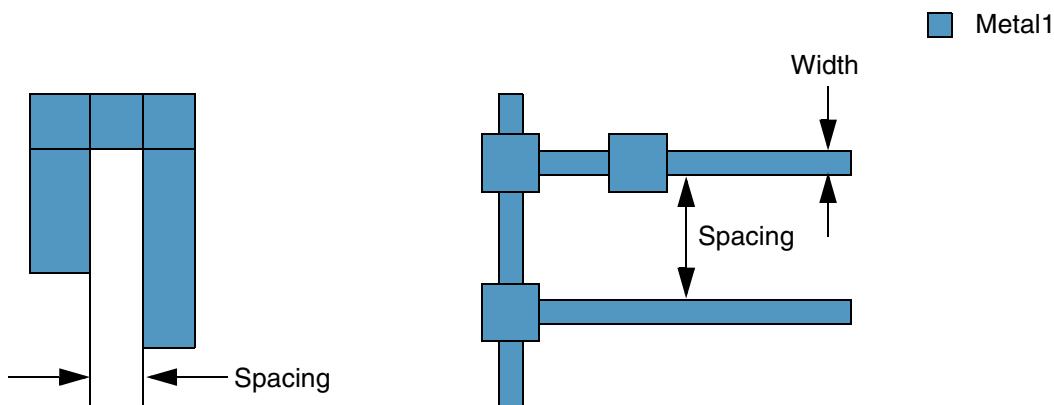
## **minSameNetSpacing (One layer)**

```
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 Constraint 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
```

## **minSideSpacing (One layer) (Advanced Nodes Only)**

```
spacings(
  ( minSideSpacing tx_layer
    [ 'shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
      | 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide
    ]
    [ 'prl f_prl | 'prlRange g_prlRange]
    [ 'sameMask]
    [ 'cornerEuclidian]
    [ 'widthRanges (g_widthRanges)]
    [ 'lengthRanges (g_lengthRanges)]
    [ 'otherWidthRanges (g_otherWidthRanges)]
    [ 'otherLengthRanges (g_otherLengthRanges)]
    [ 'horizontal | 'vertical]
    [ 'deltaVoltage f_voltage]
    [ 'exceptSpacingRanges l_exceptSpacingRanges]
    f_spacing
  )
)
;spacings
```

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.

**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

'shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide  
| 'longSideToLongSide | 'longSideToAnySide | 'anySideToAnySide

The sides between which spacing is measured.

- 'shortSideToShortSide: Spacing is measured between a short side of the first shape and a short side of the other shape.
- 'shortSideToLongSide: Spacing is measured between a short side of the first shape and a long side of the other shape.
- 'shortSideToAnySide: Spacing is measured between a short side of the first shape and any side of the other shape.
- 'longSideToLongSide: Spacing is measured between a long side of the first shape and a long side of the other shape.
- 'longSideToAnySide: Spacing is measured between a long side of the first shape and any side of the other shape.
- 'anySideToAnySide: Spacing is measured between any side of the first shape and any side of the other shape.

Type: Boolean

'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

(ICADV12.3 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.

'sameMask

(ICADV12.3 Only) The constraint applies only between shapes on the same mask.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'cornerEuclidian      (ICADV12.3 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

'widthRanges (*g\_widthRanges*)

The constraint applies only if the width of the first shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint.

'lengthRanges (*g\_lengthRanges*)

(ICADV12.3 Only) The constraint applies only if the length of the first shape falls in one of these ranges.

Type: Floating-point values specifying a range of lengths that trigger the constraint.

'otherWidthRanges (*g\_otherWidthRanges*)

The constraint applies only if the width of the other shape falls in one of these ranges.

Type: Floating-point values specifying a range of widths that trigger the constraint.

'otherLengthRanges (*g\_otherLengthRanges*)

(ICADV12.3 Only) The constraint applies only if the length of the other shape falls in one of these ranges.

Type: Floating-point values specifying a range of lengths 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

`'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.

## Examples

- [Example 1: minSideSpacing with shortSideToShortSide, prl, widthRanges, and otherWidthRanges](#)
- [Example 2: minSideSpacing with shortSideToLongSide and deltaVoltage](#)
- [Example 3: minSideSpacing with shortSideToLongSide and exceptSpacingRanges](#)

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

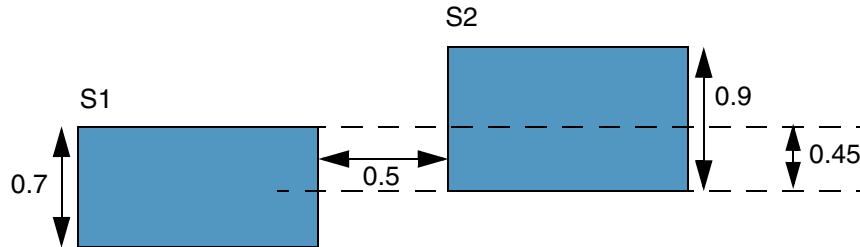
---

#### ***Example 1: minSideSpacing with shortSideToShortSide, prl, widthRanges, and otherWidthRanges***

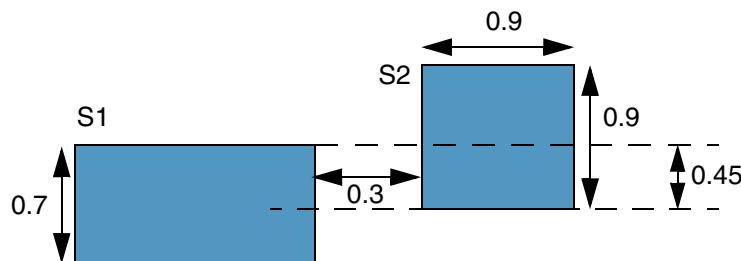
The spacing between the short sides of shapes S1 and S2 must be greater than or equal to 0.4 if the following conditions are met:

- 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 "Metall1"
    'shortSideToShortSide
    'prl 0.0
    'widthRanges "[0.6 0.8]"
    'otherWidthRanges "[0.9 1.0]"
    0.4
  ) ;spacings
```



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 Constraint Reference

### Spacing Constraints (One Layer)

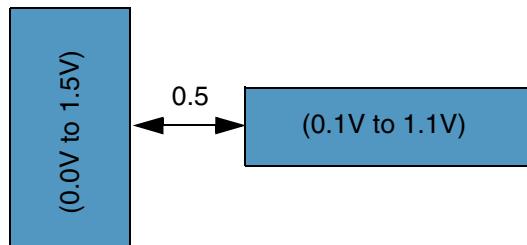
#### **Example 2: minSideSpacing with shortSideToLongSide and deltaVoltage**

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
```

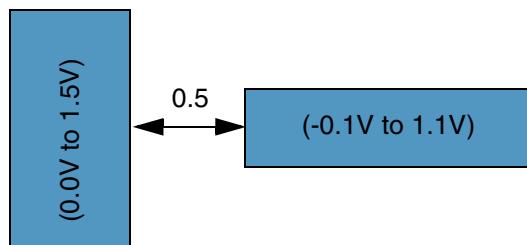
 Metal1

$$\begin{aligned} \text{Delta voltage} &= \max(\max1, \max2) - \min(\min1, \min2) \\ &= \max(1.5, 1.1) - \min(0.0, 0.1) \\ &= 1.5 - 0.0 = 1.5 \end{aligned}$$



a) The constraint does not apply because the voltage difference is only 1.5V (and not greater than 1.5V).

$$\begin{aligned} \text{Delta voltage} &= \max(\max1, \max2) - \min(\min1, \min2) \\ &= \max(1.5, 1.1) - \min(0.0, -0.1) \\ &= 1.5 - (-0.1) = 1.6 \end{aligned}$$



b) FAIL. The voltage difference is 1.6V (>1.5V). However, the spacing between the shapes is 0.5 (<0.6).

## Virtuoso Technology Data Constraint Reference

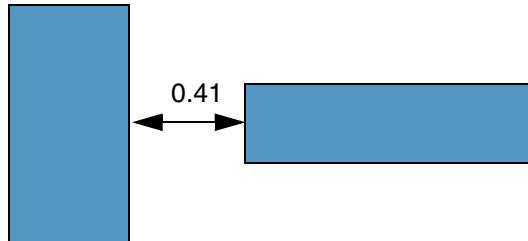
### Spacing Constraints (One Layer)

#### ***Example 3: minSideSpacing with shortSideToLongSide and exceptSpacingRanges***

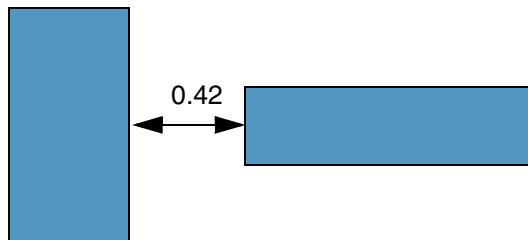
The spacing between a long side and a short side of two adjacent Metal1 shapes must be greater than or equal to 0.6. The constraint does not apply if the spacing is equal to 0.3 or greater than 0.4 and less than 0.42.

```
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.

## minSpacing (One layer)

```
spacings(
    ( minSpacing tx_layer
        ['inLayerDir tx_layer2]
        [{{'horizontal | 'vertical} ['eolWidth f_eolWidth]}
        ['sameNet ['PGNet] | 'sameMetal]
        ['sameMask]
        ['ignoreShapesInRanges (((("width")))(g_ignoreTable))]
        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} ['eolWidth f_eolWidth]}
        ['sameNet ['PGNet] | 'sameMetal]
        ['sameMask
            |
            [{{'mask1 | 'mask2 | 'mask3} ['ignoreIntermediateShapes]]
        ]
        ['exceptEolWidth f_eolWidth]
        ['ignoreShapesInRanges (((("width")))(g_ignoreTable))]
        [f_default]
    )
    (g_table) ['manhattan]
)
    ( minSpacing tx_layer
        (( "twoWidths" nil nil "length" nil nil )
        ['inLayerDir tx_layer2]
        [{{'horizontal | 'vertical} ['eolWidth f_eolWidth]}
        ['sameNet ['PGNet] | 'sameMetal]
        ['sameMask]
        ['fillConcaveCorner f_fill]
        ['ignoreShapesInRanges (((("width")))(g_ignoreTable))]
        [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

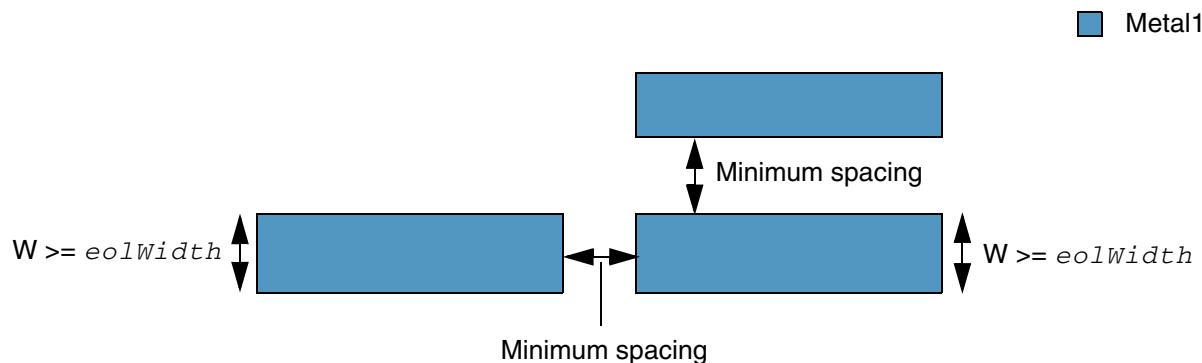
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

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.

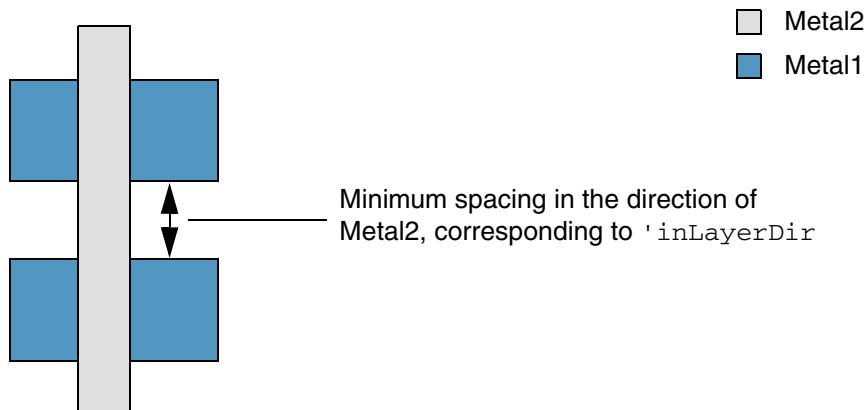
**Note:** The width of a shape is defined as the smaller of the shape's two dimensions.

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.

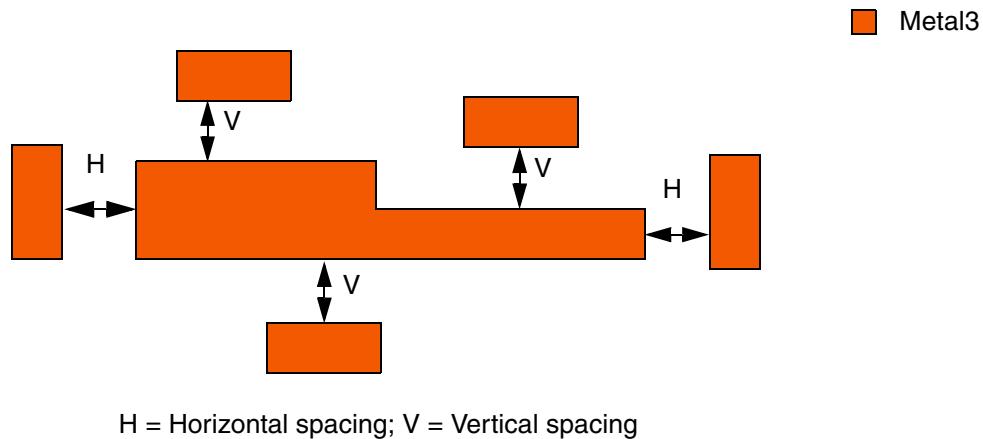
If two shapes on the specified layer are overlapped by a single shape on another layer, you can optionally specify this second layer to indicate the direction in which the constraint applies. This is useful if the structures in the design can be used at various rotations. For example, in the figure below, 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.



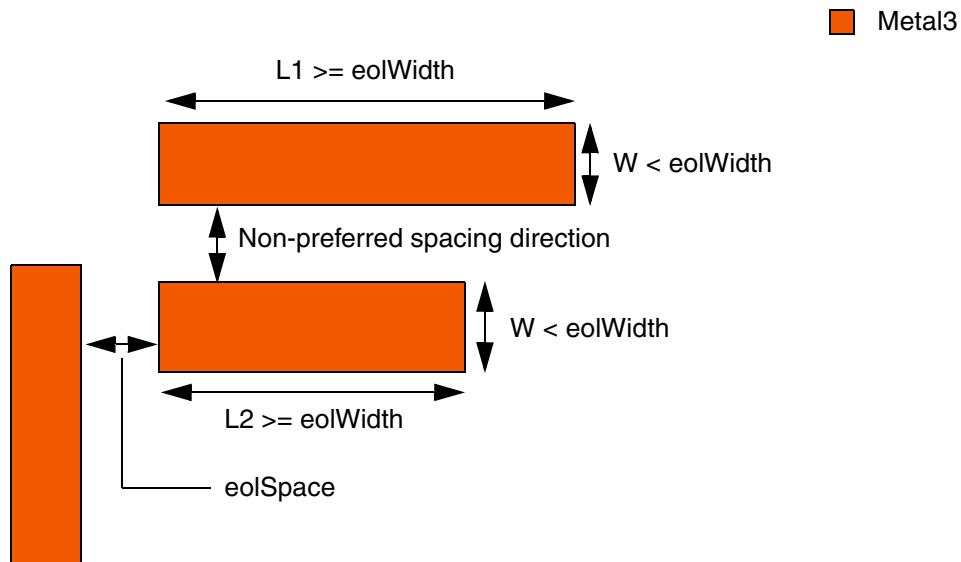
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

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.



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, `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`.



## Virtuoso Technology Data Constraint 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 required between two shapes.
"width" nil nil ["width"   "length" nil nil]	This identifies the index for <i>table</i> .
"twoWidths" nil nil "length" nil nil	This identifies the index for <i>table</i> .

*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 indices.

**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 indices (and not greater than or equal to).

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 Constraint 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

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

'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

'sameMask

(Advanced Nodes Only) The constraint applies only to shapes on the same mask.

'mask1 | 'mask2 | 'mask3

(ICADV12.3 Only) The constraint applies only to the shapes on this mask.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'ignoreIntermediateShapes

(ICADV12.3 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*) )

(Advanced Nodes Only) 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.

'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.

'exceptEolWidth *f\_eolWidth*

(Advanced Nodes Only) The constraint does not apply to edges with width less than this value.

'fillConcaveCorner *f\_fill*

(ICADV12.3 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.

*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.

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 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
```

***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 0.005, 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 Constraint Reference

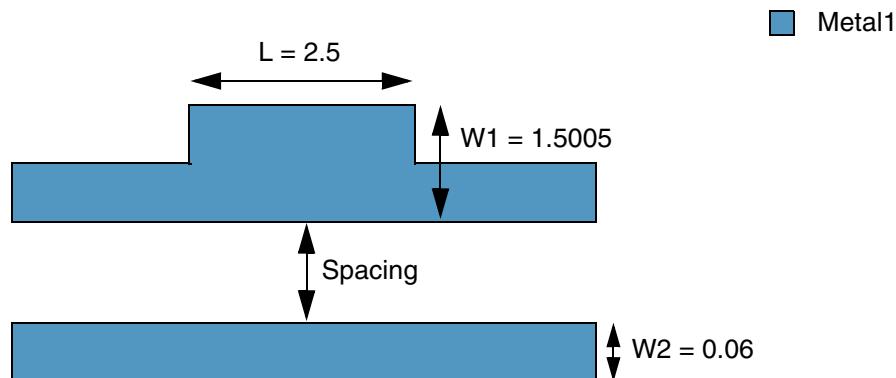
### Spacing Constraints (One Layer)

---

The following table is a representation of the example above.

<b>Spacing</b>	<b>length &gt;=</b>	0.0005	0.5605	1.5005	3.0005	7.5005
<b>width &gt;=</b>						
<b>0.0005</b>		0.12	0.18	0.50	0.90	2.50
<b>0.1805</b>		0.18	0.18	0.50	0.90	2.50
<b>1.5005</b>		0.50	0.50	0.50	0.90	2.50
<b>3.0005</b>		0.90	0.90	0.90	0.90	2.50
<b>4.5005</b>		1.50	1.50	1.50	1.50	2.50
<b>7.5005</b>		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  $W_1$  and  $L$  with the two sets of indices 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 Constraint 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.

<b>Spacing</b>	<b>length &gt;=</b>	<b>0.5</b>	<b>1.0</b>	<b>2.0</b>
<b>width &gt;=</b>				
<b>0.1</b>		0.3	0.3	0.3
<b>1.0</b>		0.3	?	0.6
<b>2.0</b>		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 Constraint 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
<b>width &gt;</b>					
<b>0.00</b>		0.15	0.20	0.50	1.00
<b>0.25</b>		0.20	0.25	0.50	1.00
<b>1.50</b>		0.50	0.50	0.60	1.00
<b>3.00</b>		1.00	1.00	1.00	1.20

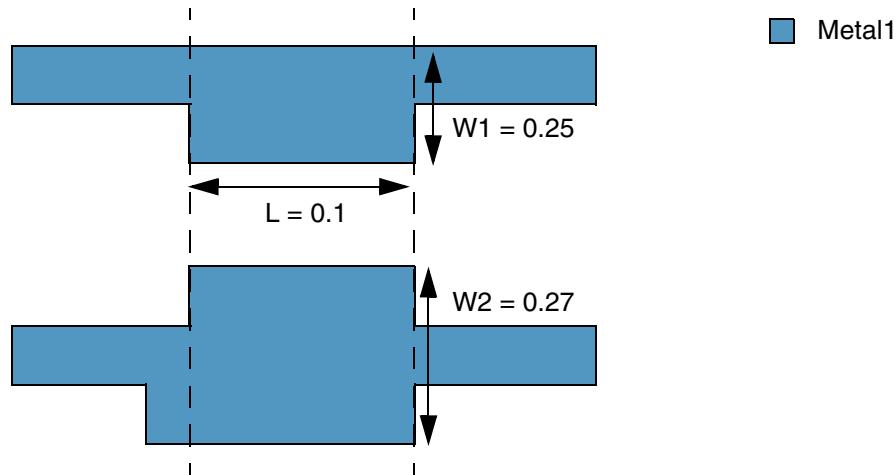
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

#### Scenario 1

In the figure below, the width of the first shape ( $W_1$ ) is 0.25 and the width of the second shape ( $W_2$ ) 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,  $W_1$  and  $L$  are looked up horizontally, and, during the second pass,  $W_2$  and  $L$  are looked up vertically. During each pass, the last width and length index pair that satisfies both " $W > \text{width}$ " and " $L > \text{length}$ " is located in the table. For example,

- $W_1$  is exactly equal to 0.25. The width index for which  $W_1 > \text{width}$  is 0.00. The value of the corresponding length index is **none**, which satisfies  $L > \text{length}$ .
- $W_2$  is equal to 0.27. The width index for which  $W_2 > \text{width}$  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:

	<b>width &gt;</b>	<b>0.00</b>	<b>0.25</b>	<b>1.50</b>	<b>3.00</b>	
	<b>length &gt;</b>	<b>none</b>	<b>0.00</b>	<b>1.50</b>	<b>3.00</b>	
<b>width &gt;</b>	<b>length &gt;</b>					
<b>0.00</b>	<b>none</b>	0.15	0.20	0.50	1.00	<b><i>W1/L</i></b>
<b>0.25</b>	<b>0.00</b>	0.20	0.25	0.50	1.00	
<b>1.50</b>	<b>1.50</b>	0.50	0.50	0.60	1.00	
<b>3.00</b>	<b>3.00</b>	1.00	1.00	1.00	1.20	
			<b><i>W2/L</i></b>			

## Virtuoso Technology Data Constraint 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:

		<b>width &gt;</b>	<b>0.00</b>	0.25	1.50	3.00
		<b>length &gt;</b>	<b>none</b>	<b>0.00</b>	<b>1.50</b>	<b>3.00</b>
<b>width &gt;</b>	<b>length &gt;</b>					
<b>0.00</b>	<b>none</b>		0.15	0.20	0.50	1.00
<b>0.25</b>	<b>0.00</b>		0.20	0.25	0.50	1.00
<b>1.50</b>	<b>1.50</b>		0.50	0.50	0.60	1.00
<b>3.00</b>	<b>3.00</b>		1.00	1.00	1.00	1.20
			<b>W1/L</b>			

### 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  $W1 > \text{width}$  is 0.25. The value of the corresponding `length` index is 0.00, which satisfies  $L > \text{length}$ .
- W2 is equal to 1.60. The `width` index for which  $W2 > \text{width}$  is 1.50. However, the value of the corresponding `length` index is 1.50, which does not satisfy  $L > \text{length}$ . Therefore, `width` index 0.25 is considered (because its corresponding `length` index of 0.00 satisfies  $L > \text{length}$ ).

## Virtuoso Technology Data Constraint 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:

		<b>width &gt; 0.00</b>	<b>0.25</b>	1.50	3.00	
		<b>length &gt; none</b>	<b>0.00</b>	1.50	3.00	
<b>width &gt; length &gt;</b>						
<b>0.00</b>	<b>none</b>	0.15	0.20	0.50	1.00	
<b>0.25</b>	<b>0.00</b>	0.20	0.25	0.50	1.00	<b>W1/L</b>
<b>1.50</b>	<b>1.50</b>	0.50	0.50	0.60	1.00	
<b>3.00</b>	<b>3.00</b>	1.00	1.00	1.00	1.20	
			<b>W2/L</b>			

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:

		<b>width &gt; 0.00</b>	<b>0.25</b>	1.50	3.00	
		<b>length &gt; none</b>	<b>0.00</b>	1.50	3.00	
<b>width &gt; length &gt;</b>						
<b>0.00</b>	<b>none</b>	0.15	0.20	0.50	1.00	
<b>0.25</b>	<b>0.00</b>	0.20	0.25	0.50	1.00	<b>W2/L</b>
<b>1.50</b>	<b>1.50</b>	0.50	0.50	0.60	1.00	
<b>3.00</b>	<b>3.00</b>	1.00	1.00	1.00	1.20	
			<b>W1/L</b>			

**Example 7: minSpacing with ignoreShapesInRanges**

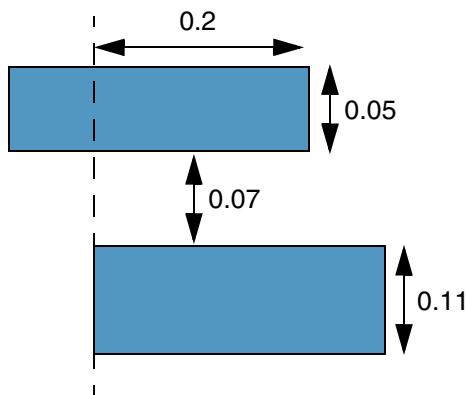
The spacing requirement can vary based on the width of adjacent shapes and the parallel run length between them. For example, the spacing between shapes must be at least 0.14 if the width of the wider of the two shapes is greater than or equal to 0.1 and the parallel run length between the shapes is 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.

```

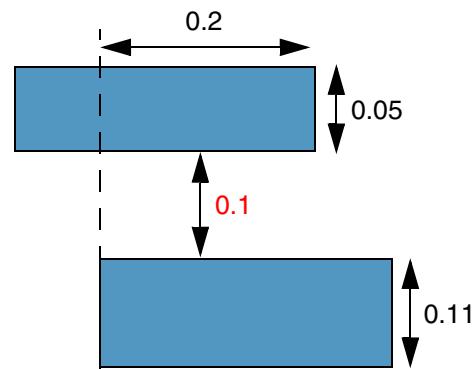
spacingTables(
  ( minSpacing "Metal1"
    (( "width" nil nil "length" nil nil ))
    'ignoreShapesInRanges ((( "width" )) (0.1 ("[0.07 0.10]")))
  )
  (
    (0.0 -0.041) 0.05
    (0.0 -0.040) 0.06
    (0.0 0.080) 0.06
    (0.1 -0.041) 0.05
    (0.1 -0.040) 0.06
    (0.1 0.080) 0.14
  )
)
) ;spacingTables

```

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.

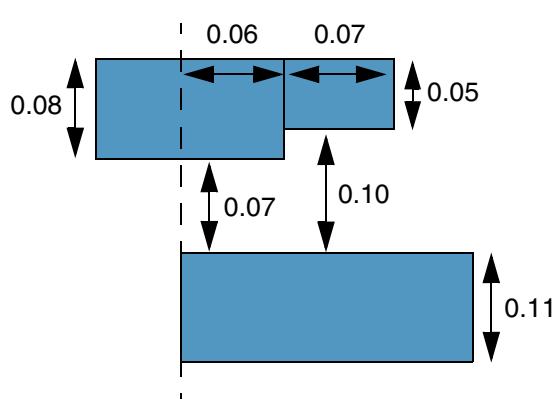


b) FAIL. The spacing between the two shapes lies outside the exemption range and is also less than 0.14 — the spacing required when the width of the wider of the two shapes is 0.11 and the parallel run length between the shapes is greater than or equal to 0.08.

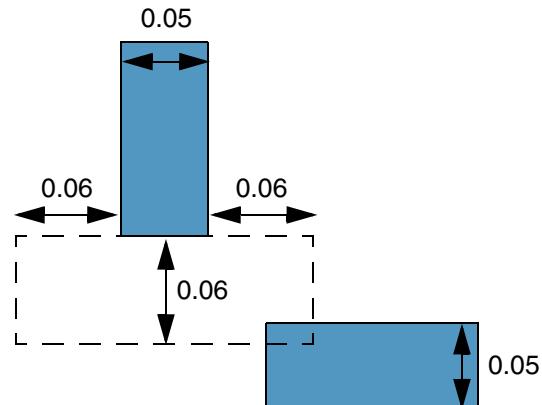
## Virtuoso Technology Data Constraint Reference

### 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.



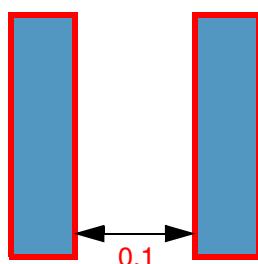
d) FAIL. The required spacing is 0.06 (width  $\geq$  -0.04). However, the neighboring shape is within a distance of 0.06. (To satisfy the spacing requirement of 0.06, it must be outside the search window.)

### ***Example 8: minSpacing with sameMask***

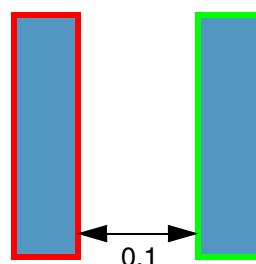
The minimum spacing between two adjacent shapes must be 0.15 if the two shapes are on the same mask, and 0.1 otherwise.

```
spacings(
  ( minSpacing "Metall1"
    0.1
  )
  ( minSpacing "Metall1"
    '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 Constraint Reference

### Spacing Constraints (One Layer)

---

#### **Example 9: minSpacing with fillConcaveCorner**

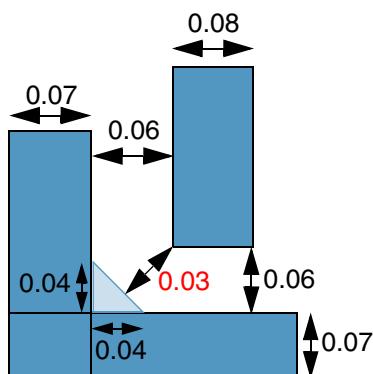
All concave corners must be filled by a triangle whose sides along the concave edges are equal to 0.04 before minimum spacing is checked. The minimum spacing requirement between shapes is 0.06, which also applies to the concave-corner fill. The spacing requirement can vary based on the width of adjacent shapes and is determined by looking up the spacing table. However, this non-minimum spacing requirement does not apply to the concave fill.

```

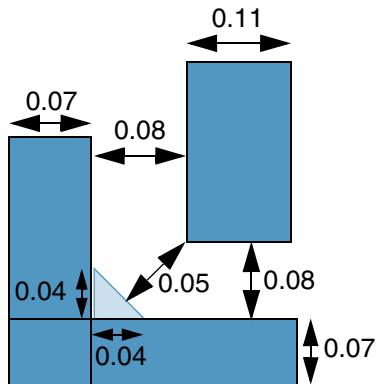
spacingTables(
  ( minSpacing "Metal1"
    (( "twoWidths" nil nil "length" nil nil )
     'fillConcaveCorner 0.04
    )
    (
      (0.0 0.0) 0.06
      (0.0 0.1) 0.08
      (0.1 0.0) 0.08
      (0.1 0.1) 0.09
    )
  )
);spacingTables

```

■ 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 Constraint Reference

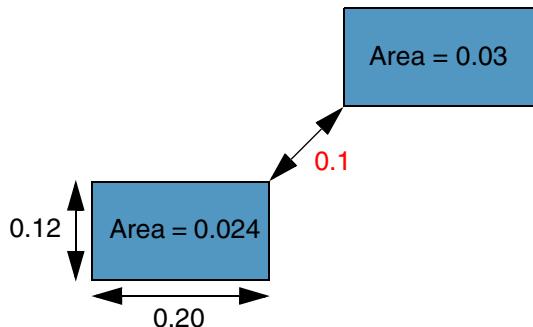
### Spacing Constraints (One Layer)

#### Example 10: *minSpacing with area*

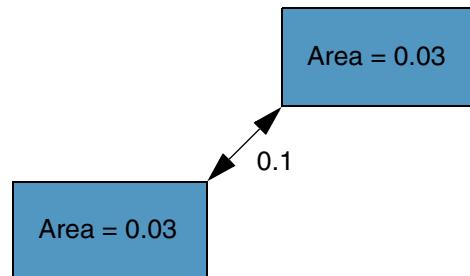
The minimum spacing between two adjacent shapes must be 0.15 if at least one of the shapes has area less than 0.03.

```
spacings(
  ( minSpacing "Metall1"
    '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 Constraint Reference

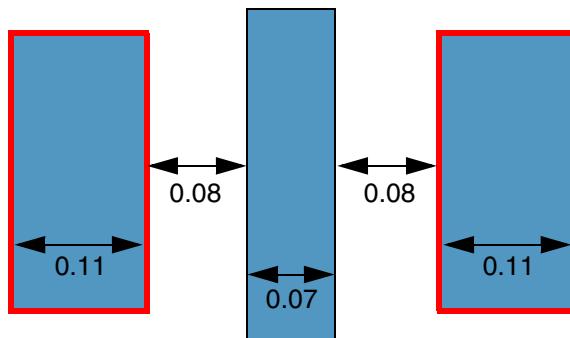
### Spacing Constraints (One Layer)

#### **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).

## minSpanLengthSpacing (Advanced Nodes Only)

```
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]
         ['minSpanSpacingRanges g_minSpanTable
             ['minSpanSpacingRangesPrl g_minSpanPrlTable]
         ]
         ['minSpanForExceptEolWidth f_minSpanEolWidth]
     )
     (g_table)
   )
) ;spacingTables
```

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.

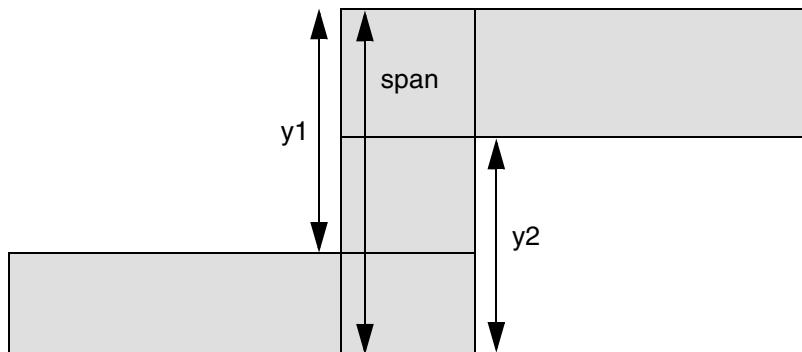
## Virtuoso Technology Data Constraint Reference

### 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  $length$ .

## Values

*tx\_layer*      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 *table*.

*g\_table*      The format of the table is as follows:

(*f\_prl f\_spanLength*) *f\_spacing*

where,

- *f\_prl* applies if the parallel run length between the two shapes is greater than or equal to this value.
- *f\_spanLength* applies if the larger of the two spans is greater than or equal to this value.
- *f\_spacing* 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 Constraint 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

(ICADV12.3 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 Constraint 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*

(ICADV12.3 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*

(ICADV12.3 Only) The spacing for shapes in the same span length range can be exactly 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.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

' sameMask

(ICADV12.3 Only) The constraint applies only to shapes on the same mask.

This constraint must be enforced between shapes on the same mask, regardless of any different-mask shapes that may be placed between them.

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 Constraint Reference

### Spacing Constraints (One Layer)

---

`'minSpanSpacingRangesPrl g_minSpanPrlTable`

(ICADV12.3 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_prl`

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.
- $f_{prl}$  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`

(ICADV12.3 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.

## 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 minSpanSpacingRangesPrl](#)
- [Example 6: minSpanLengthSpacing with eolWidth and eolSpacingTable](#)

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

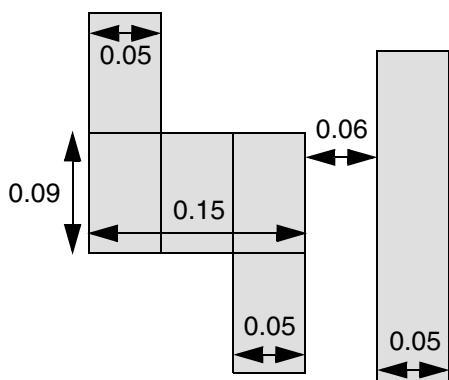
#### **Example 1: minSpanLengthSpacing with horizontal, exceptJogLength, and exactSpacingTable**

- 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.

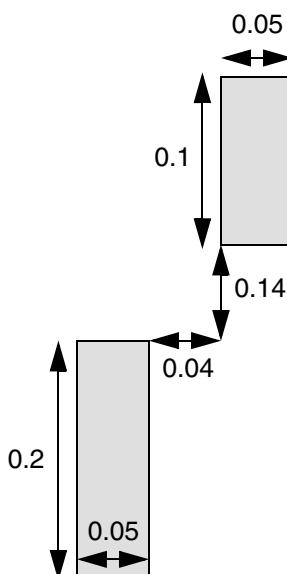
```

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

```



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).

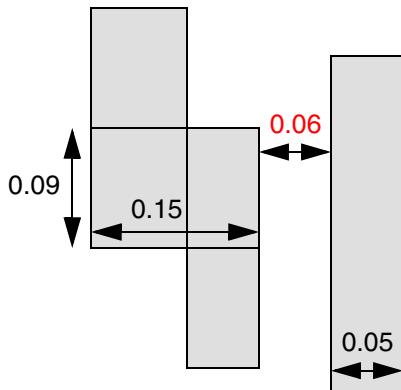


b) The constraint does not apply if the parallel run length is less than zero.

## Virtuoso Technology Data Constraint 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  $\geq 0.14$ .

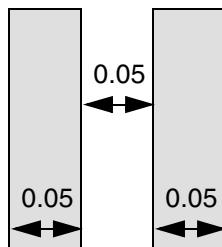
### ***Example 2: minSpanLengthSpacing with horizontal and exactSpacingTable***

The minimum horizontal spacing between two wires must be as follows:

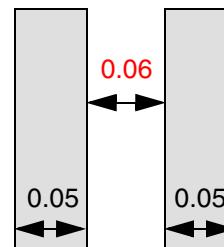
- If the larger of the two spans is less than 0.1, the spacing must be at least 0.07. An exact spacing of 0.05 is also acceptable.
- If the larger of the two spans is greater than or equal to 0.1, the spacing must be at least 0.11.

```
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 exactly equal to 0.05 or greater than or equal to 0.07. A spacing of 0.06 is not allowed.

***Example 3: minSpanLengthSpacing with horizontal, exactSpacingTable, and spacingToMinSpanTable***

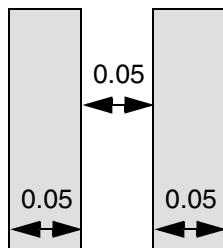
The minimum horizontal spacing requirement between two wires is as follows:

- For wires with span less than or equal to 0.09, the spacing must be at least 0.07 or exactly 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.

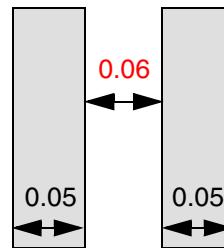
```
spacingTables ( □ Metal2  
  ( minSpanLengthSpacing "Metal2"  
    ( ( "prl" nil nil "spanlength" nil nil )  
      'horizontal  
      'sameMask  
      'exactSpacingTable ((( "spanlength" ))  
        (  
          0.00 0.05  
          0.09 0.08  
          0.11 0.09  
        )  
      )  
      '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 Constraint Reference

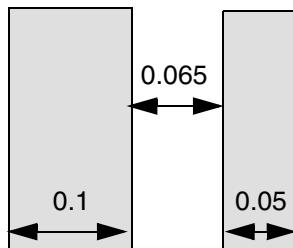
### Spacing Constraints (One Layer)



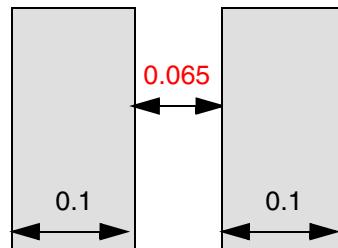
a) PASS. An exact spacing of 0.05 is allowed.



b) FAIL. The spacing between wires must be exactly 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.  
$$S1 < S2 < S3 < \dots < SN$$
- 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 Constraint Reference

### Spacing Constraints (One Layer)

---

The following table represents the constraint's value table. Note that 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, note that 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).

<b>spanlength/ prl</b>	<b>-prl1 &lt; 0</b>	<b>prl = 0</b>	<b>prl2 &gt; 0</b>
<b>Sp1</b>	S12	S12	S12
<b>Sp2</b>	S12	S12	S12
<b>Sp3</b>	0	S4	S4
<b>Sp4</b>	0	S7	S7
<b>Sp5</b>	0	S10	S10
<b>Sp6</b>	0	S10	S10
<b>Sp7</b>	0	S10	S10
<b>Sp8</b>	0	S13	S10
<b>Sp9</b>	0	S11	S14

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

The table below represents the spacing table for the 'minSpanSpacingRanges' parameter. Note that 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.

<b>spanLength/ spanLength</b>	<b>Sp1</b>	<b>Sp2</b>	<b>Sp3</b>	<b>Sp4</b>	<b>Sp5</b>	<b>Sp6</b>
<b>Sp1</b>	=X1, =X2, = X3, >=S12	=X1, =X2, =X3, >=S12	>=S9	>=S3	>=S8	>=S7
<b>Sp2</b>	=X1, =X2, =X3, >=S12	=X1, =X2, =X3, >=S12	>=S9	>=S3	>=S8	>=S7
<b>Sp3</b>	>=S9	>=S9	0	0	0	0
<b>Sp4</b>	>=S3	>=S3	0	0	0	0
<b>Sp5</b>	>=S8	>=S8	0	0	0	0
<b>Sp6</b>	>=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 Constraint Reference

### Spacing Constraints (One Layer)

---

```

spacingTables(
    minSpanLengthSpacing "Metal1"
        ("prl" 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")
        ))
    )
    (
        (-prl1 sp1) S12
        (0 sp1) S12
        (prl2 sp1) S12
        (-prl1 sp2) S12
        (0 sp2) S12
        (prl2 sp2) S12
        (0 sp3) S4
        (prl2 sp3) S4
        (0 sp4) S7
        (prl2 sp4) S7
        (0 sp5) S10
        (prl2 sp5) S10
        (0 sp6) S10
        (prl2 sp6) S10
        (0 sp7) S10
        (prl2 sp7) S10
        (0 sp8) S13
        (prl2 sp8) S10
        (0 sp9) S11
        (prl2 sp9) S14
    )
)
; spacingTables

```

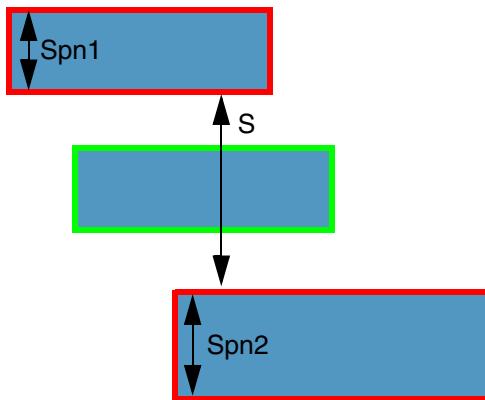
- █ Metal1
- █ mask1
- █ mask2

## Virtuoso Technology Data Constraint Reference

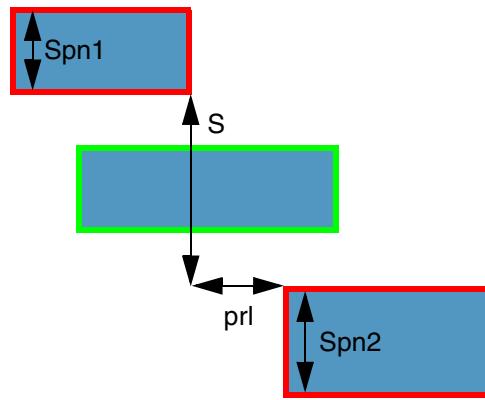
### Spacing Constraints (One Layer)

---

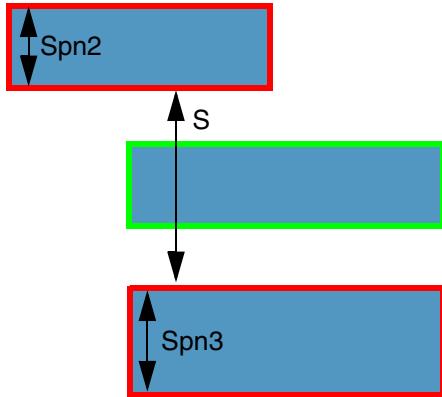
**Figures (a), (b), (c), and (d) illustrate how the `'minSpanSpacingRanges` parameter is used to specify a set of spacing ranges for wires with smaller spans.**



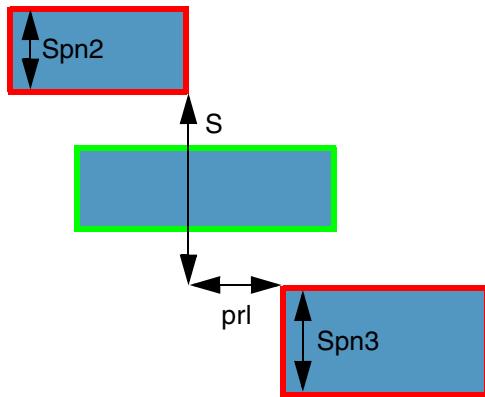
a)  $\text{prl} > 0$ ; fails if  $S \neq X1, X2, \text{ or } X3$  and is  $< S12$ ; passes if  $S \geq S12$ .



b)  $\text{prl} < 0$ ; the constraint does not apply if  $\text{prl} < -\text{prl1}$ ; fails if  $\text{prl} \geq -\text{prl1}$  and  $S \neq X1, X2, \text{ or } X3$  and is  $< S12$ .



c)  $\text{prl} > 0$ ; fails if  $S < S4$ ; passes if  $S \geq S4$



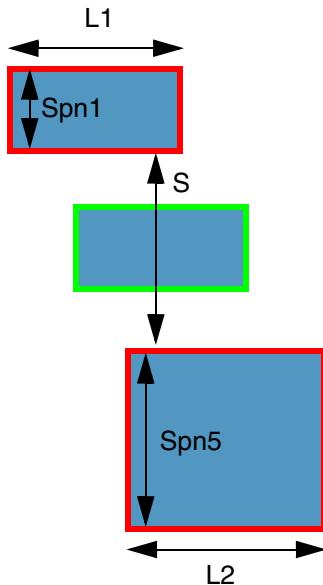
d)  $\text{prl} < 0$ ; passes even if  $S < S4$ .

## Virtuoso Technology Data Constraint Reference

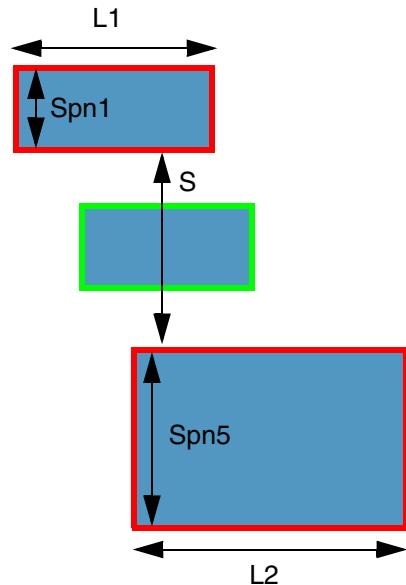
### Spacing Constraints (One Layer)

---

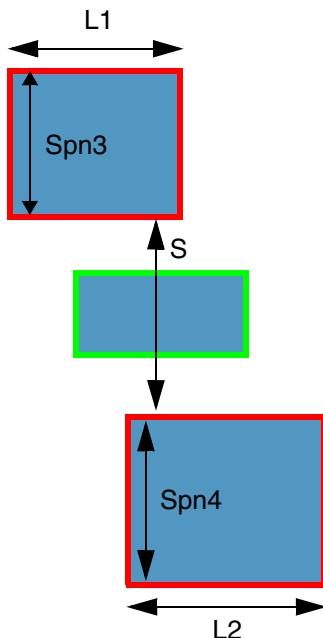
**Figures (e), (f), and (g) illustrate how spacing is calculated for wider span wires by using both 'eolWidth' and 'minSpanForExceptEolWidth' parameters.**



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 >= S8 in the minSpanSpacingRanges table for Spn5/Spn1. Therefore, if S >= 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 >= S7, the constraint is met.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

#### **Example 4b: minSpanLengthSpacing with minSpanSpacingRanges**

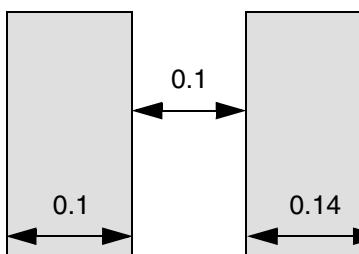
The minimum spacing requirement between two wires is as follows:

- 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.

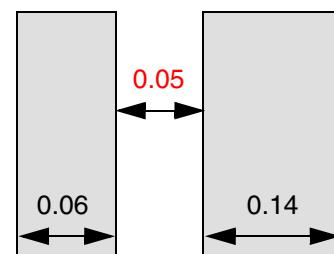
```

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

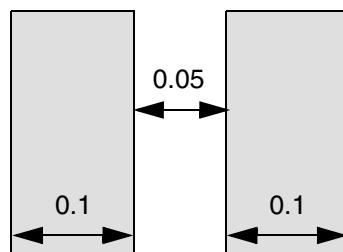
```



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.

**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:

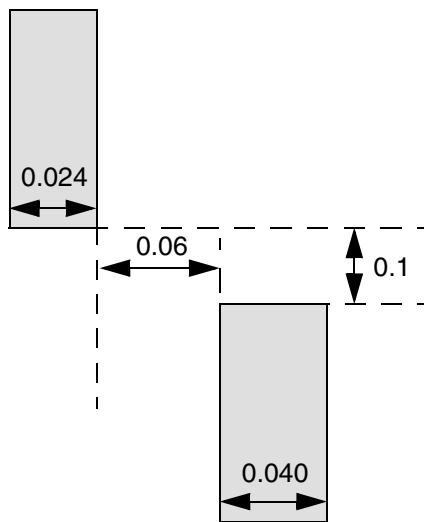
- If one span is greater than or equal to 0.0235 and the other span is greater than or equal to 0.0395, the spacing between the wires must be exactly equal to 0.06 or greater than 0.1.
- If the larger of the two spans is less than 0.0395, the spacing must be greater than or equal to 0.12.
- If the larger 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"
    ( ( "prl" 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
```

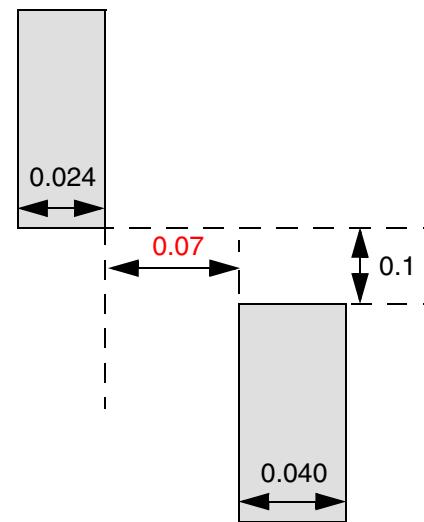
Metal2

## Virtuoso Technology Data Constraint Reference

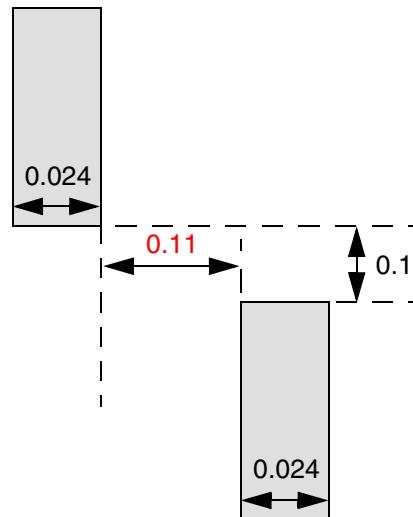
### Spacing Constraints (One Layer)



a) PASS. An exact spacing of 0.06 is allowed.



b) FAIL. The spacing between the wires must be exactly 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.

***Example 5: minSpanLengthSpacing with minSpanSpacingRanges and minSpanSpacingRangesPrl***

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 exactly 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 exactly 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 exactly 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"
    (( "prl" 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")
       )
     )
   )
   'minSpanSpacingRangesPrl
   (
     (( "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
```

***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"  
    ( ( "prl" 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

```

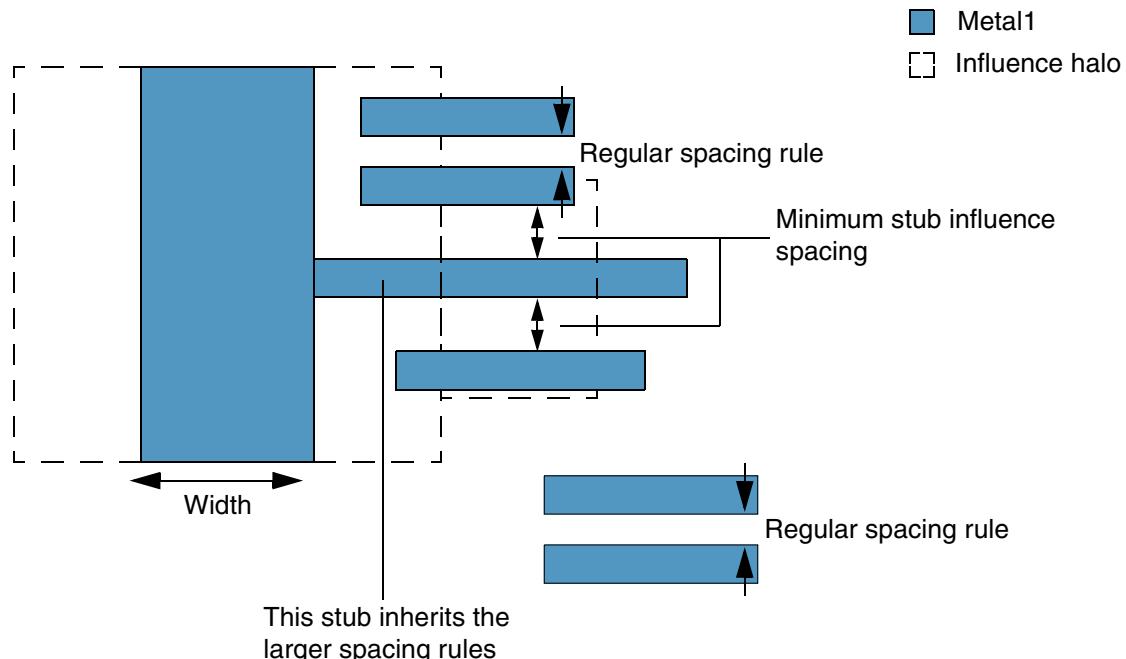
spacings(
    ( minStubInfluenceSpacing tx_layer
        'width f_stubWidth
        f_spacing
    )
)
;spacings

spacingTables(
    ( minStubInfluenceSpacing tx_layer
        'width f_stubWidth
        (( "width" nil nil ["distance" nil nil] )
        [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 Constraint 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> . The format of a row in a 1-D <i>table</i> is as follows: $(f\_width \ f\_spacing)$ 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: $(f\_width \ f\_distance) \ f\_spacing$ where, <ul style="list-style-type: none"><li>■ <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.</li><li>■ <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.</li></ul> 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

*'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
```

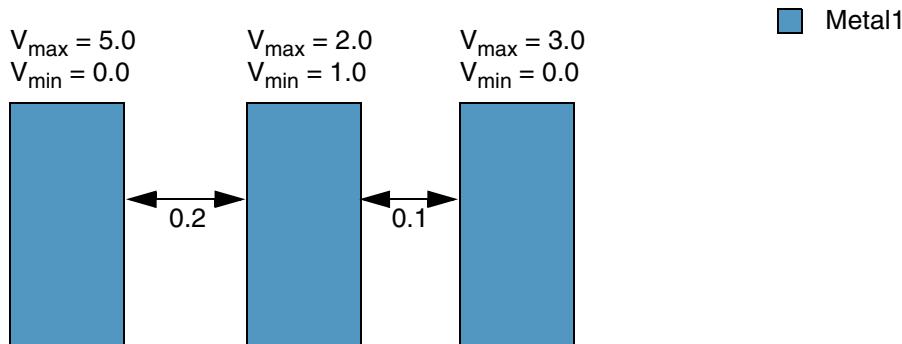
## minVoltageSpacing (One layer)

```
spacingTables(
  ( minVoltageSpacing tx_layer
    ( "voltage" nil nil )
    (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.



**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\_layer*      The layer on which the constraint is applied.  
Type: String (layer name) or Integer (layer number)

"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 wires.  
The actual voltage difference 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

None

## 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

```
spacingTables(  
    ( minVoltageSpacing "Metal1"  
        (( "voltage" nil nil ))  
        (  
            0.0  0.065  
            1.5  0.085  
            3.3  0.105  
        )  
    )  
) ;spacingTables
```

## **oneSideSpacing (ICADV12.3 Only)**

```
spacings(
  ( oneSideSpacing (tx_layer1 tx_layer2 ... tx_layerN)
  )
) ; spacings
```

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 Constraint 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
```

## **pgViaTrack (Advanced Nodes Only)**

```
spacings(
  ( pgViaTrack tx_cutLayer
    'coreOffset f_offset
    {'horizontalTrack | 'verticalTrack}
    f_pitch
  )
) ;spacings
```

Specifies the pitch of the tracks on *cutLayer*, in the specified direction.

### **Values**

*tx\_cutLayer*      The cut layer on which the constraint is applied.

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

### **Parameters**

'coreOffset *f\_offset*

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 Constraint Reference

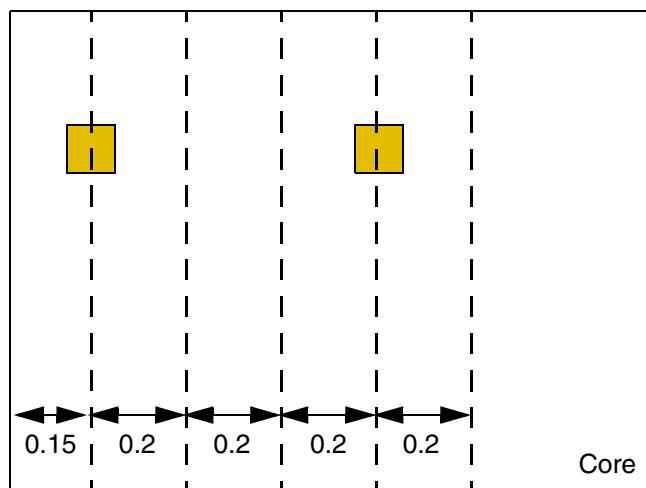
### Spacing Constraints (One Layer)

#### 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.

## **shapeRequiredBetweenSpacing (Advanced Nodes Only)**

```
orderedSpacings(  
    ( shapeRequiredBetweenSpacing tx_layer1 tx_layer2  
        ['horizontal | 'vertical]  
        f_spacing  
    )  
) ;orderedSpacings
```

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**

'horizontal   'vertical	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 Constraint Reference

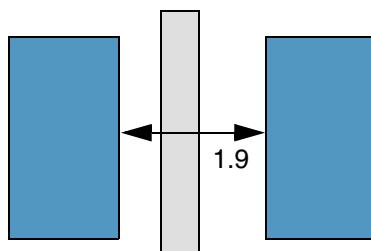
### Spacing Constraints (One Layer)

#### Example

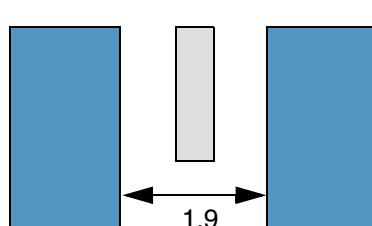
Two Metal1 shapes that are less than 2.0 apart in the horizontal direction must have a Metal2 shape between them, and this Metal2 shape must fully separate the two Metal1 shapes.

```
orderedSpacings(  
    ( shapeRequiredBetweenSpacing "Metal1" "Metal2"  
        'horizontal  
        2.0  
    )  
) ;orderedSpacings
```

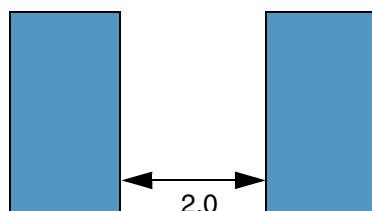
 Metal2  
 Metal1



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 exactly equal to 2.0.

### **snapPatternDefOffset (ICADV12.3 Only)**

```
spacings(
    ( snapPatternDefOffset
        { 'horizontal' | 'vertical' | 'snapPatternDef' t_name}
        f_offset
    )
) ;spacings
```

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.

## 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
```

## **trimMinAdjacentSpacing (ICADV12.3 Only)**

```
spacings(
  ( trimMinAdjacentSpacing tx_layer
    'count x_count
    'distance f_distance
    [ 'mask1 | 'mask2 | 'mask3 | 'mask4]
    f_spacing
  )
) ; spacings
```

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**

'count <i>x_count</i>	The constraint applies only if the number of adjacent trim shapes is greater than or equal to this value.
'distance <i>f_distance</i>	The distance between two trim shapes must be less than this value for them to be considered adjacent.
'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

## Virtuoso Technology Data Constraint Reference

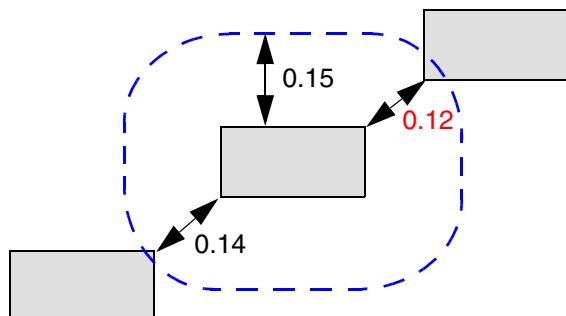
### Spacing Constraints (One Layer)

#### Example

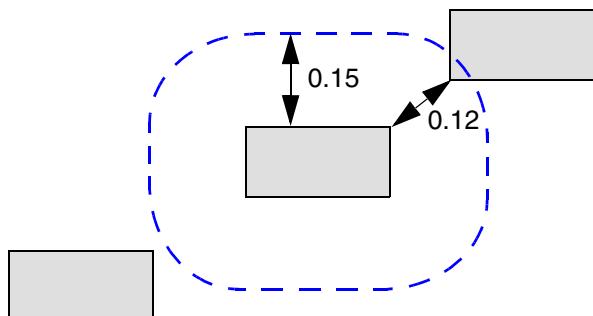
The distance between adjacent trim shapes must be at least 0.13 if a trim shape has at least two trim shapes at a distance less than 0.15 from it.

```
spacings(
  ( trimMinAdjacentSpacing "TrimMetal1"
    'count 2
    'distance 0.15
    0.13
  )
) ;spacings
```

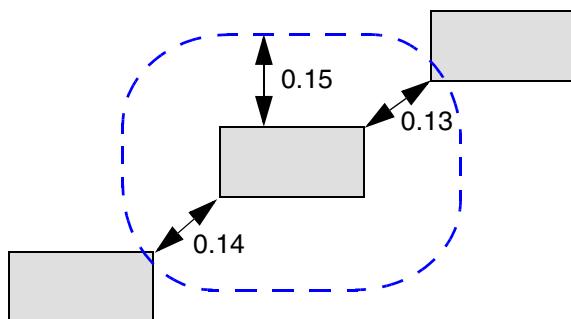
 TrimMetal1



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.

## **trimMinSpacing (One layer) (ICADV12.3 Only)**

```
spacings(
    ( trimMinSpacing tx_layer
        [ '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_layer1 tx_layer2 ... tx_layerN)
            | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ]
        [ 'sameMask ]
        f_spacing
    )
)
;spacings
```

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.

### **Parameters**

'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*.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

'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*

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.

'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)

'sameMask

The constraint applies only between shapes on the same mask.

Type: Boolean

## Virtuoso Technology Data Constraint 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 (*tx\_layer1 tx\_layer2 ... tx\_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 (*tx\_layer1 tx\_layer2 ... tx\_layerN*).

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

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

### 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 exactly 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 Constraint Reference

### Spacing Constraints (One Layer)

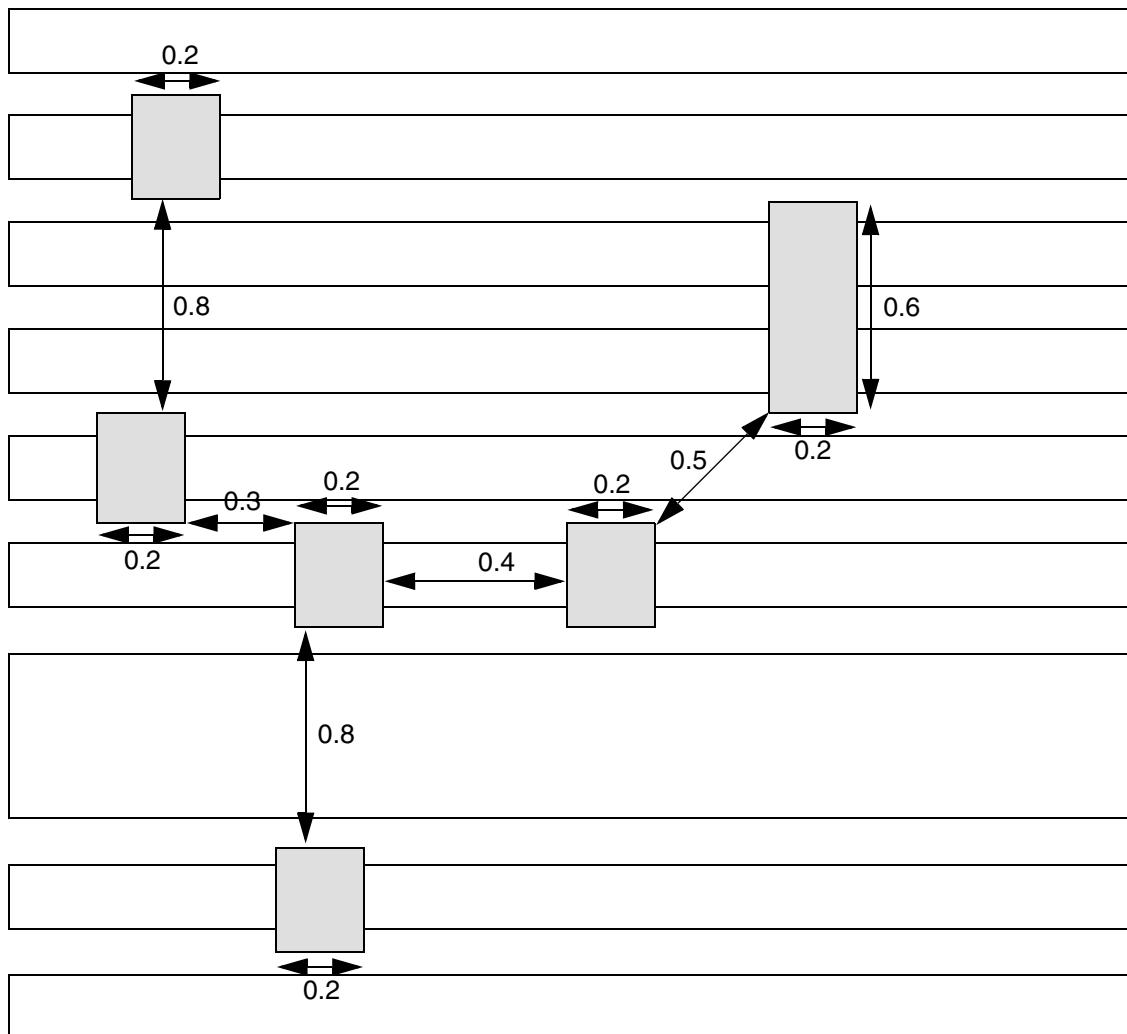
---

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(                                     □ TrimMetal1
  ( trimShape "TrimMetal1"
    'exactWidth 0.2
    'maxLength 0.6
    "midTrack"
  )
  ( trimMinSpacing "TrimMetal1"
    'prlSpacing (0.3 0.4)
    'endToEndSpacing 0.8
    'vertical
    0.5
  )
) ;spacings
```

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)



PASS. The width of all TrimMetal1 shapes is exactly 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 Constraint Reference

### Spacing Constraints (One Layer)

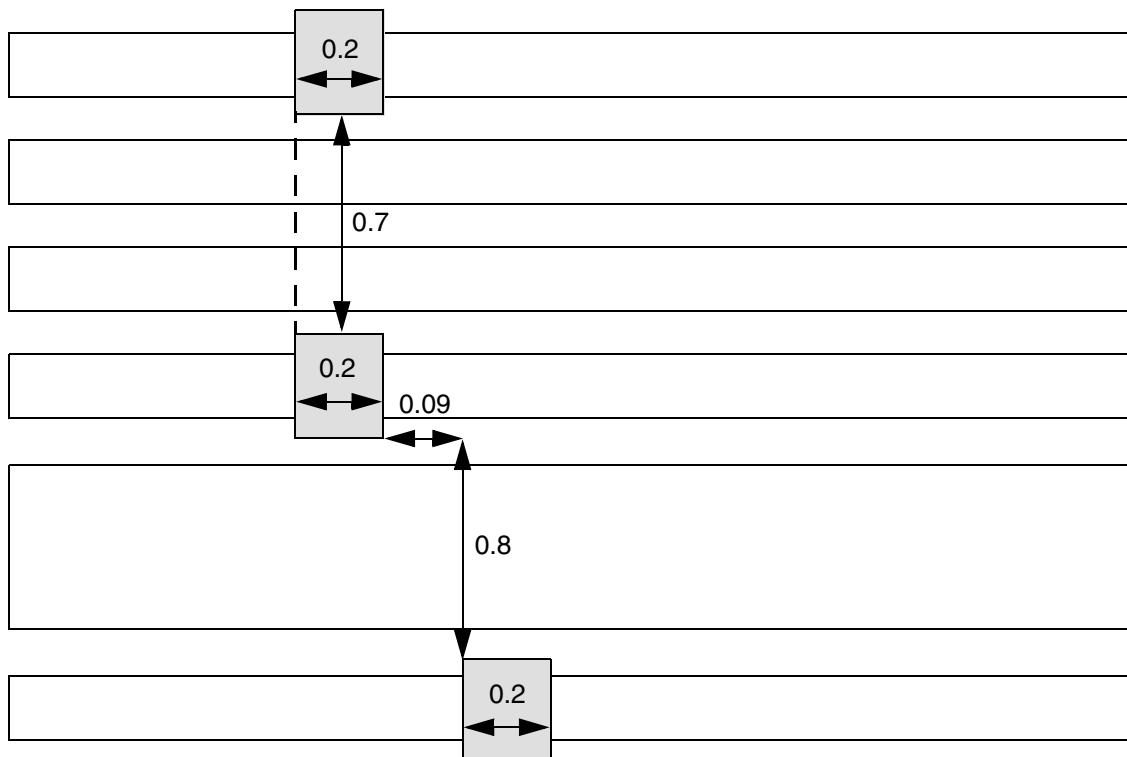
---

```

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

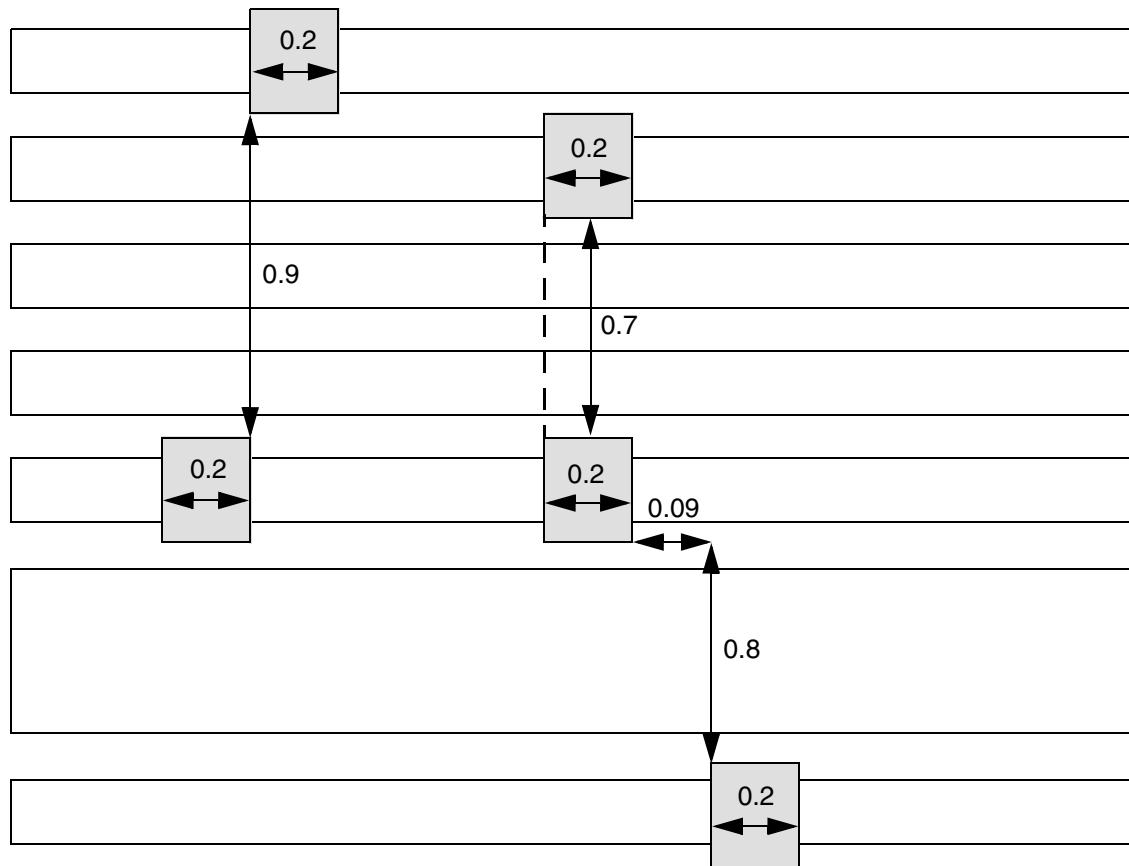
## Virtuoso Technology Data Constraint Reference

### 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 Constraint Reference

### Spacing Constraints (One Layer)

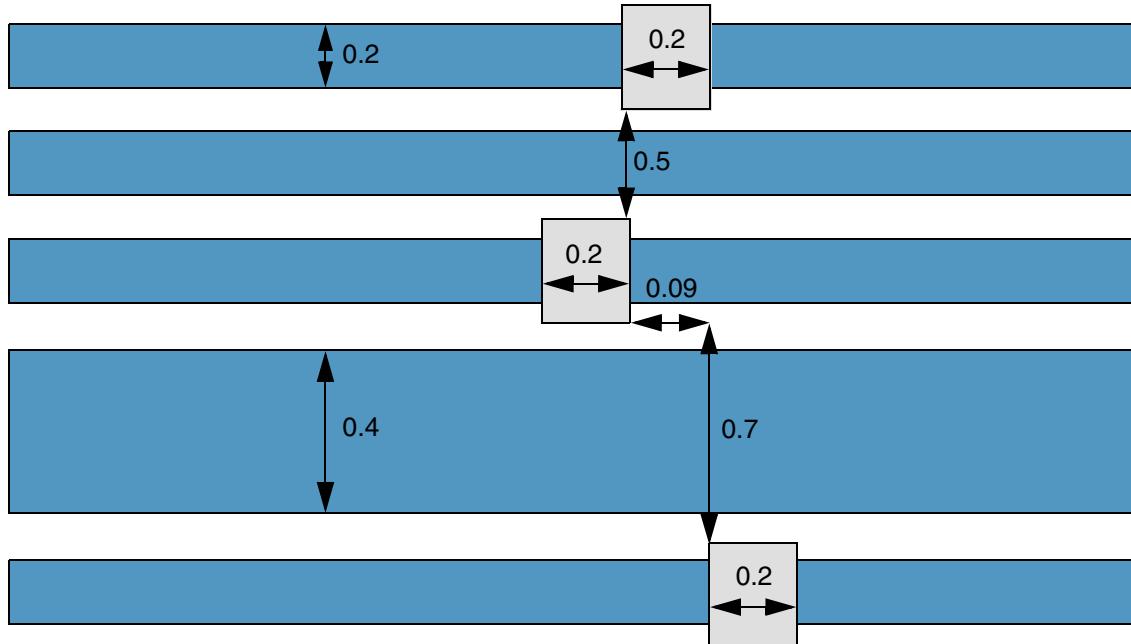
---

#### **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.

***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 (ICADV12.3 Only)**

```

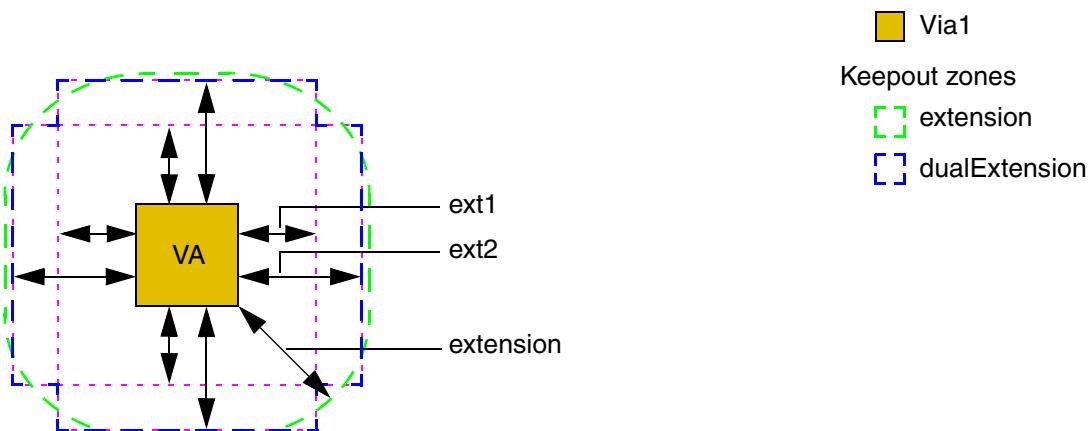
spacings(
    ( viaKeepoutZone tx_cutLayer
        'cutClass {f_width | (f_width f_length) | t_name}
        ['otherCutClass {f_width1 | (f_width1 f_length1) | t_name1}]
        'dualExtension (f_ext1 f_ext2)
        ['dualSideExtension (f_sideExt1 f_sideExt2)]
        ['sameMask]
        ['exactAligned f_spacing
            ['allEdge | 'shortEdge | 'longEdge]
        ]
        ['insideLayers (tx_layer1 tx_layer2 ... tx_layerN)
            | 'outsideLayers (tx_layer1 tx_layer2 ... tx_layerN)
        ]
        f_extension
    )
)
;spacings

```

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.

Additional keepout zones are defined by using '`dualExtension` (can be specified for both square and rectangular cut classes) and, optionally, '`dualSideExtension` (can be specified only for rectangular cut classes).

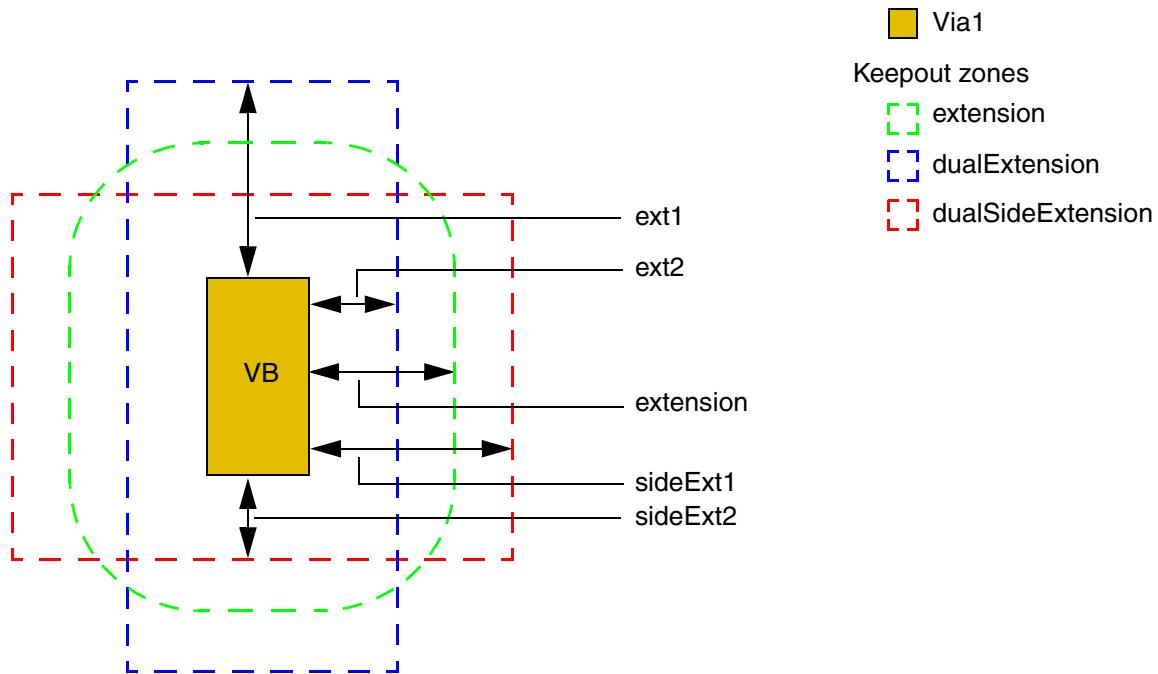
The following figure illustrates the two keepout zones for a square via cut, one defined by the constraint value (green) and the other defined by '`dualExtension` (blue). When only '`dualExtension` is specified for a square or rectangular via cut, the keepout zone is the union of the two regions formed by extending outward the two extension values, `ext1` and `ext2`, from each pair of opposite edges of the via cut, as shown below.



## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

The following figure illustrates the three keepout zones for a rectangular via cut when 'dualSideExtension' is also specified: the first defined by the constraint value (green), the second defined by 'dualExtension' (blue, with *ext1* on ends and *ext2* on sides), and the third defined by 'dualSideExtension' (red, with *sideExt1* on sides and *sideExt2* on ends).



## 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_extension</i>	All cut edges are extended outward by this value in Euclidean distance to define a keepout zone.

## Virtuoso Technology Data Constraint 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.

'dualSideExtension (*f\_sideExt1 f\_sideExt2*)

The two values together define a third keepout zone that extends outward by *sideExt1* from both sides of the via cut and by *sideExt2* from both ends of the via cut.

**Note:** The parameter applies only to rectangular via cuts.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (One Layer)

---

' 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

'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 (*tx\_layer1 tx\_layer2 ... tx\_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 (*tx\_layer1 tx\_layer2 ... tx\_layerN*).

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

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

## Virtuoso Technology Data Constraint Reference

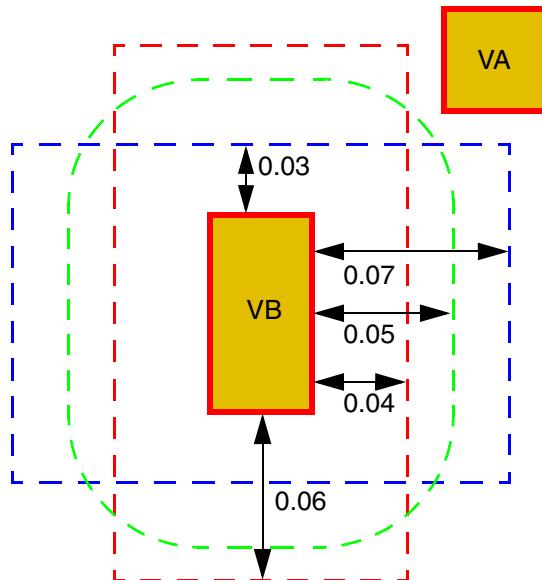
### Spacing Constraints (One Layer)

#### Example

A VA via cut on layer Via1 must not overlap with the keepout zones defined for a VB via cut on layer Via1 if both via cuts are on the same mask.

```
spacings(
  ( viaKeepoutZone "Via1"
    'cutClass "VB"
    'otherCutClass "VA"
    'dualExtension (0.03 0.07)
    'dualSideExtension (0.04 0.06)
    'sameMask
    0.05
  )
) ;spacings
```

 Via1  
 Mask1  
Keepout zones  
 extension  
 dualExtension  
 dualSideExtension



---

## Spacing Constraints (Two Layers)

---

This chapter includes the following constraints:

- [allowedSpacingRanges \(Two layers\)](#)
- [maxSpacing](#)
- [minCenterLineSpacing \(Two layers\) \(Advanced Nodes Only\)](#)
- [minClusterSpacing \(Two layers\) \(Advanced Nodes Only\)](#)
- [minCornerSpacing \(Two layers\) \(Advanced Nodes Only\)](#)
- [minCutRoutingSpacing \(Two layers\)](#)
- [minInnerVertexSpacing \(Two layers\)](#)
- [minInnerVertexSpacing \(Three layers\)](#)
- [minNeighboringShapesSpacing \(Advanced Nodes Only\)](#)
- [minSameNetSpacing \(Two layers\)](#)
- [minSideSpacing \(Two layers\) \(Advanced Nodes Only\)](#)
- [minSpacing \(Two layers\)](#)
- [minSpacingOver](#)
- [minTouchingDirSpacing](#)
- [minVoltageSpacing \(Two layers\)](#)
- [shapeRequiredSpacing](#)
- [trimMinSpacing \(Two layers\) \(ICADV12.3 Only\)](#)

## 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

*tx\_layer1*

The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

*g\_ranges*

The allowed spacing ranges are specified as follows:

(*spacingRange1* *spacingRange2* ...)

Type: Floating-point values specifying the spacing ranges that are allowed.

## Virtuoso Technology Data Constraint 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.

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 greater than 0.3 and less than or equal to 0.5, and the spacing between a shape on Gate2 and a shape on Poly2 must be greater than 0.2 and less than 0.4.

```
spacings(
  ( allowedSpacingRanges "Gate1" "Poly1"
    ("(0.3 0.5]")
  )
  ( allowedSpacingRanges "Gate2" "Poly2"
    ("(0.2 0.4)")
  )
) ;spacings
```

## 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.

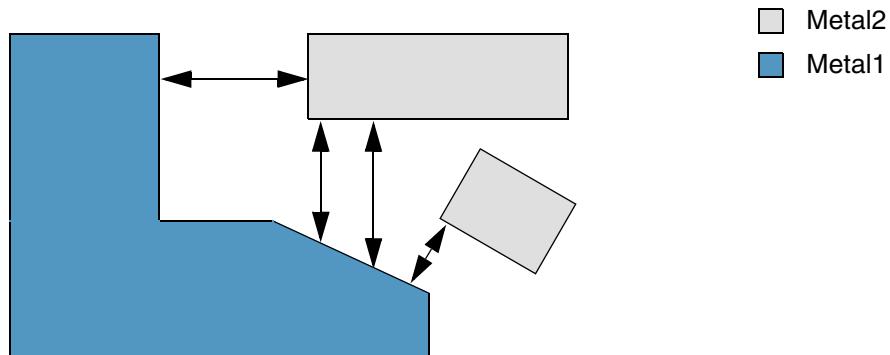
```
spacingTables(  
  ( allowedSpacingRanges "Poly" "Active"  
    (( "width" nil nil ))  
    (  
      0.1  (0.12 ">=0.15")  
      0.2  (0.15 "[0.2 0.21]" ">=0.3")  
    )  
  )  
) ;spacingTables
```

## 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

*tx\_layer*                          The layer on which the constraint is applied.

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

*f\_spacing*                          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) (Advanced Nodes Only)**

```
orderedSpacings(  
  ( minCenterLineSpacing tx_layer1 tx_layer2  
    f_spacing  
  )  
) ; orderedSpacings
```

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 Constraint Reference

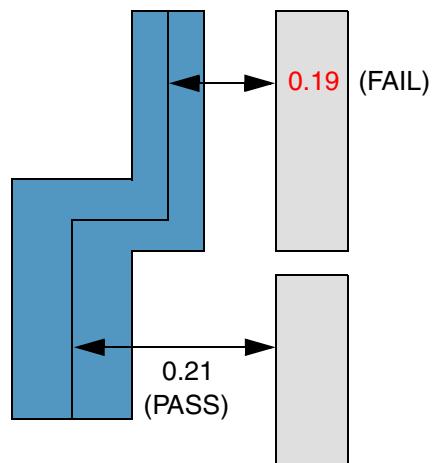
### Spacing Constraints (Two Layers)

#### 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) (Advanced Nodes Only)**

```
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
```

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**

*tx\_layer1*

The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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 Constraint 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

'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 Constraint Reference

### Spacing Constraints (Two Layers)

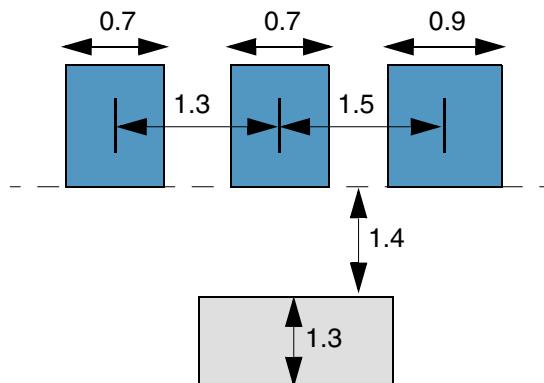
#### **Example 1: minClusterSpacing with widthRange, numShapesRange, spacingRange, and otherWidthRange**

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
```



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 Constraint 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:

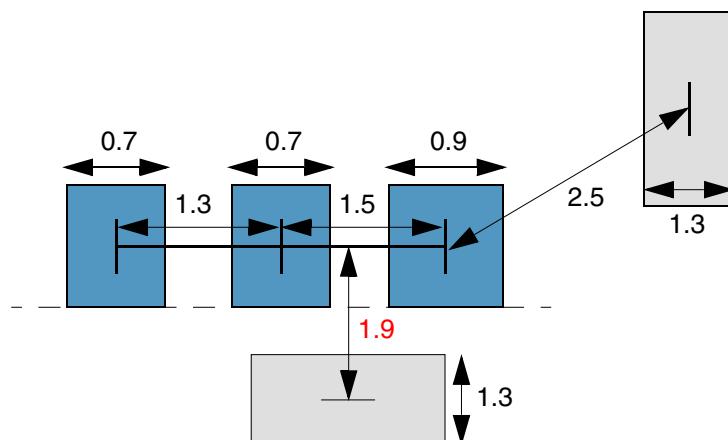
- The width of each shape in the cluster must be in the range [0.7 0.9].
- The number of shapes in the cluster must be at least 3 and at most 4.
- The spacing measured between the centerlines of neighboring shapes in the cluster must be in the range [1.3 1.5].

```

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 Constraint Reference

### Spacing Constraints (Two Layers)

#### ***Example 3: minClusterSpacing with widthRange, numShapesRange, spacingRange, centerLineDistance, otherWidthRange, and groupVertical***

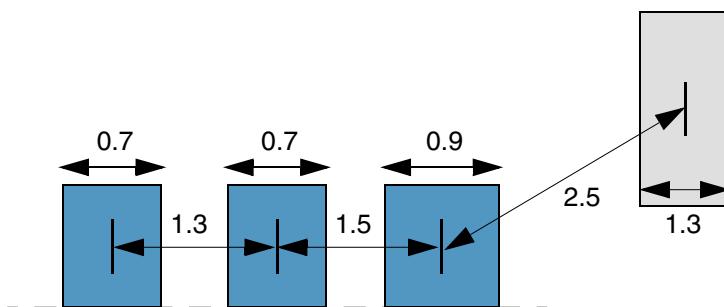
The spacing measured from the centerline of a vertical cluster to the centerline of a neighboring Metal2 shape must be at least 2.0 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5.

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.

**Example 4: *minClusterSpacing* with *widthRange*, *numShapesRange*, *spacingRange*, *horizontal*, *otherWidthRange*, and *groupVertical***

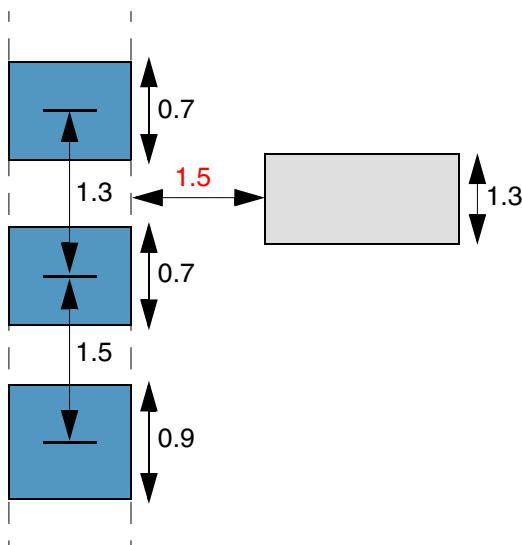
The horizontal spacing between a vertical cluster and a neighboring Metal2 shape must be at least 1.7 if the width of the neighboring shape is greater than or equal to 1.2 and less than or equal to 1.5. The shapes in the cluster must satisfy the following conditions:

- 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].

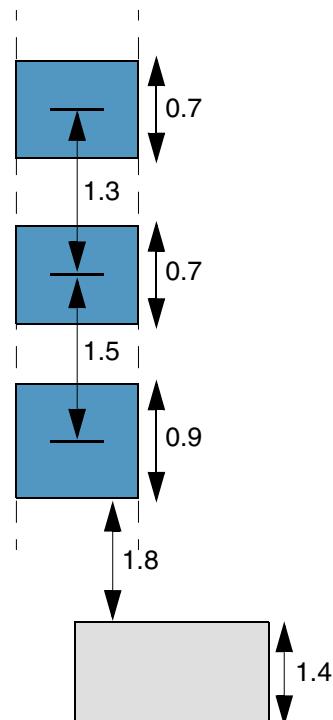
orderedSpacings(

```
( 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.

## **minCornerSpacing (Two layers) (Advanced Nodes Only)**

```
orderedSpacings(
  ( minCornerSpacing tx_layer1 tx_layer2
    ['allCorner | 'concaveCorner | 'convexCorner]
    ['otherAllCorner | 'otherConcaveCorner | 'otherConvexCorner]
    ['deltaVoltage f_deltav]
    g_spacing
  )
) ; orderedSpacings
```

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 f_deltav	The constraint applies if the maximum voltage difference between the two shapes is greater than this value.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

---

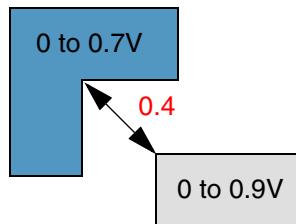
#### Example

The spacing between Metal1 and Metal2 corners must be greater than or equal to 0.5 if the following conditions are met:

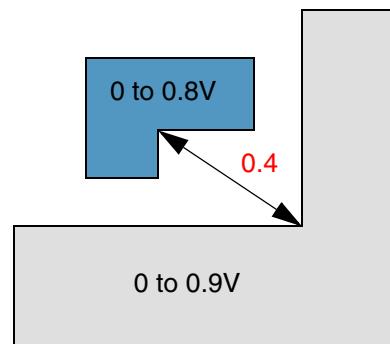
- The Metal1 corner is concave.
- The Metal2 corner is concave or convex.
- The maximum voltage difference between Metal1 and Metal2 shapes is greater than 0.8V.

```
orderedSpacings(
    ( minCornerSpacing "Metal1" "Metal2"
        'concaveCorner
        'otherAllCorner
        'deltaVoltage 0.8
        0.5
    )
) ;orderedSpacings
```

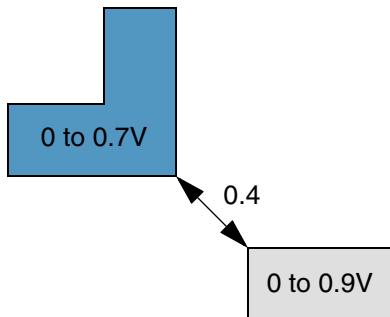
Metal2  
 Metal1



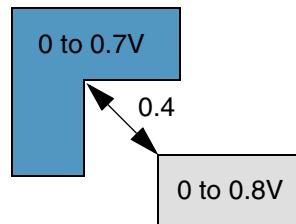
a) FAIL. The maximum voltage difference is 0.9 (>0.8), but the spacing is only 0.4 (<0.5).



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).

## minCutRoutingSpacing (Two layers)

```
orderedSpacings(
    ( minCutRoutingSpacing tx_layer1 tx_layer2
        ['cutClass {f_width | (f_width f_length) | t_name}
         ['exceptSameNet ['shortEdge]] | ['toConcaveCorner ['prl f_pr1]]
        ]
        ['enclosingLayer tx_enclLayer
         ['viaEdgeType x_viaEdgeType]
         ['enclosingLayerWidth f_enclWidth
          ['anyOppositeExtension f_oppExtension]
         ]
        ]
        ['parallelEdgeWithin f_parWithin 'otherExtension f_otherExtension]
        ['parallelExtension f_parExtension]
        ['maskOverlap]
        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 *prl* with the via cut.
- The spacing from a cut edge with zero extension to the below-metal shape is exactly equal to (*parWithin* - 0.001) and to the above-metal shape is less than *parWithin*.

## Virtuoso Technology Data Constraint 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 (routing layer) on which the constraint is applied. Type: String (layer name) or Integer (layer number)
<i>f_spacing</i>	The minimum required spacing.

#### 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 <a href="#">cutClasses</a> constraint).
■ <i>f_width</i> : Width	
■ <i>f_length</i> : Length	
■ <i>t_name</i> : Name of the cut class	
'exceptSameNet	The constraint does not apply to shapes on the same net. Type: Boolean
'shortEdge	The constraint applies only between a metal edge and the short edge of a rectangular cut shape. Type: Boolean
'toConcaveCorner	The constraint applies between a cut shape and a concave corner of the enclosing metal shape. Type: Boolean
'prl <i>f_value</i>	(Advanced Nodes Only) 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. Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

---

'enclosingLayer *tx\_enclLayer*

(Advanced Nodes Only) The enclosing layer for  
'viaEdgeType extension checks.

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

'viaEdgeType *x\_viaEdgeType*

(Advanced Nodes Only) The constraint applies only to the cut edges of this type (the type is defined in the viaEdgeType constraint).

'enclosingLayerWidth *f\_enclWidth*

(Advanced Nodes Only) 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*

(Advanced Nodes Only) The constraint applies only if a cut shape has extensions less than this value on any two opposite sides.

'parallelEdgeWithin *f\_parWithin*

(Advanced Nodes Only) The constraint applies only if the neighboring shape on the metal layer below is at a distance exactly 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*

(Advanced Nodes Only) 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.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

---

'parallelExtension *f\_parExtension*

(Advanced Nodes Only) 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`.

'maskOverlap

(Advanced Nodes Only) The constraint applies between a cut shape and the edge of an overlap between two metal shapes on two different masks.

Type: Boolean

## 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](#)

## Virtuoso Technology Data Constraint Reference

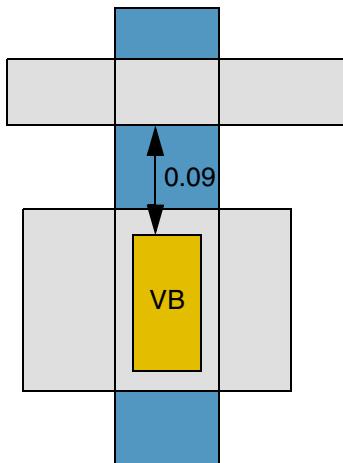
### Spacing Constraints (Two Layers)

#### **Example 1: minCutRoutingSpacing with cutClass, exceptSameNet and shortEdge**

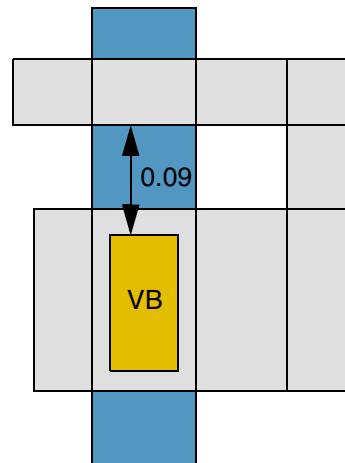
The spacing between the short edge of a Via1 via cut of type VB (0.4x0.6) and a Metal2 shape must be at least 0.1 if the via cut and the Metal2 wire are not on the same net.

```
orderedSpacings(  
    ( minCutRoutingSpacing "Via1" "Metal2"  
        'cutClass "VB"  
        'exceptSameNet 'shortEdge  
        0.1  
    )  
) ;orderedSpacings
```

[Light Gray Box]	Metal2
[Yellow Box]	Via1
[Blue Box]	Metal1



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 Constraint Reference

### Spacing Constraints (Two Layers)

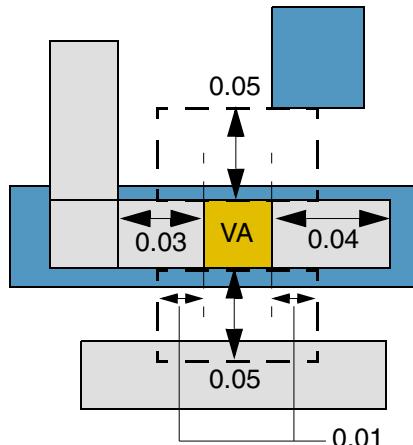
---

#### ***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 exactly 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
```

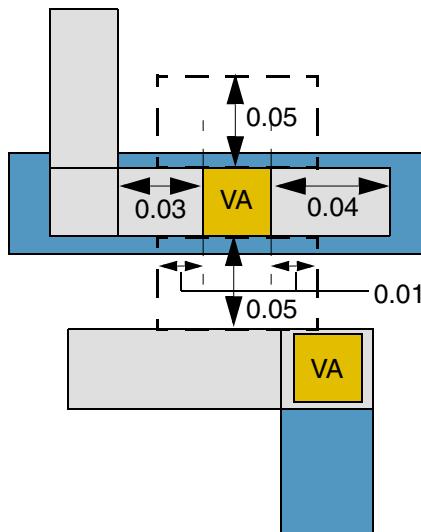


a) FAIL. The via cut is of type VA and has zero extensions on two opposite sides and extensions less than 0.06 on the other two opposite sides. The via cut also has two neighboring wires on opposite sides—the sides with zero extension—with parallel run length greater than -0.1 with the via cut. The distance of the below-metal (Metal1) wire is exactly 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).

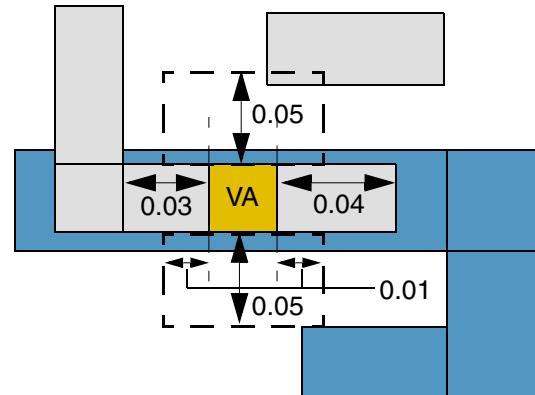
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

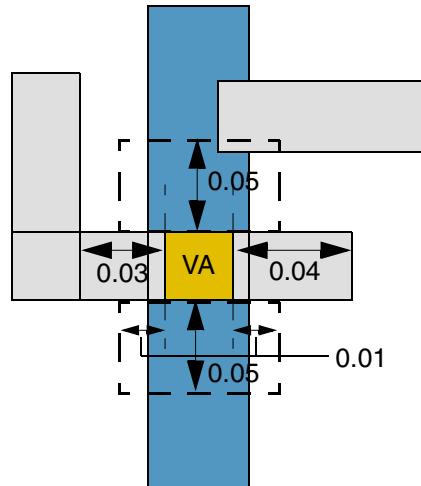
---



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 Constraint Reference

### Spacing Constraints (Two Layers)

---

#### ***Example 3: minCutRoutingSpacing with enclosingLayer, viaEdgeType, and enclosingLayerWidth***

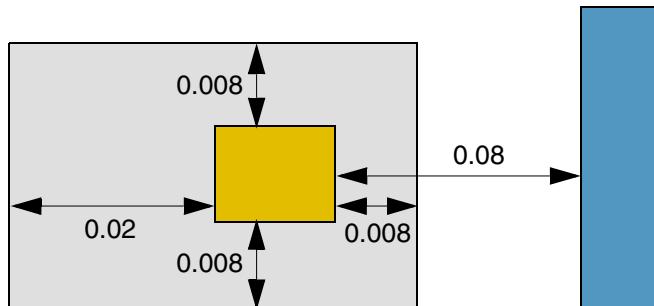
The spacing between a Via1 via cut and a neighboring Metal1 wire must be at least 0.08. The constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.009.

```

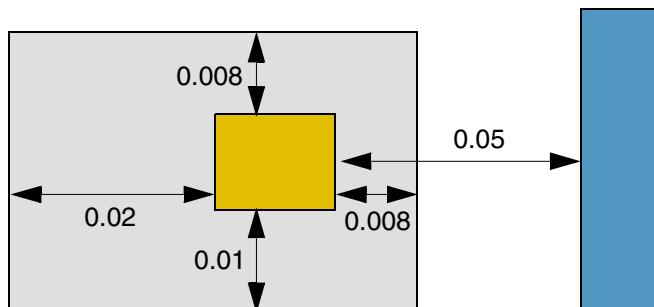
spacings(
    ( viaEdgeType "Via1"
        'anyOppositeExtension 0.009
        'negateAnyOppositeExtension
        1
    )
) ;spacings
orderedSpacings(
    ( minCutRoutingSpacing "Via1" "Metal1"
        'enclosingLayer "Metal2"
        'viaEdgeType 1
        'enclosingLayerWidth 0.0
        0.08
    )
) ;orderedSpacings

```

	Metal2
	Via1
	Metal1



- 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 has have extension 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).

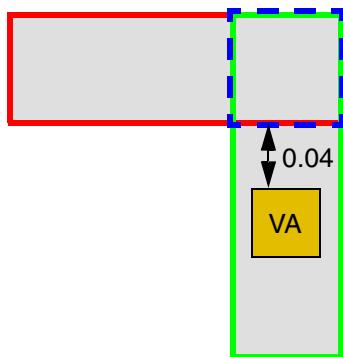
## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

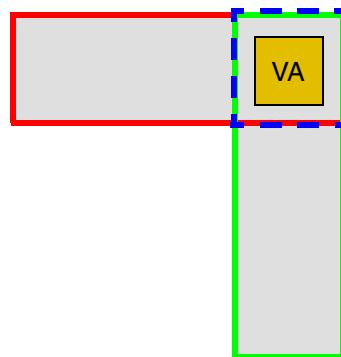
#### **Example 4: minCutRoutingSpacing with cutClass and maskOverlap**

The spacing between a Via1 via 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.

## **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 *layer2* edges adjacent to the *layer1* inside corner are at a distance greater than or equal to this value from the *layer1* edges that form the inside corner.

## Virtuoso Technology Data Constraint 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.

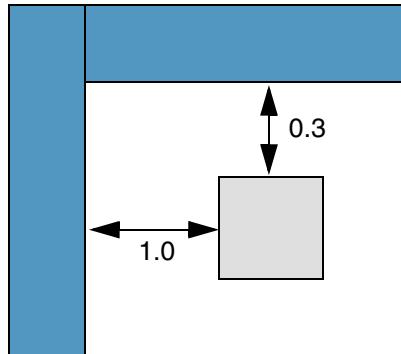
```

spacings(
  ( minSpacing "Metal1" "Metal2"
    0.3
  )
) ;spacings

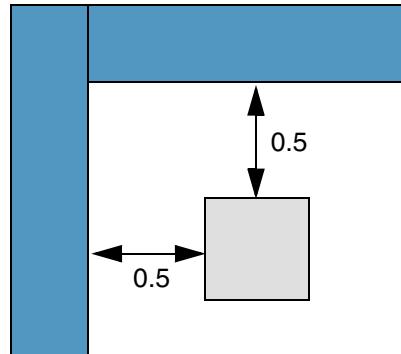
orderedSpacings(
  ( minInnerVertexSpacing "Metal1" "Metal2"
    'distance 0.5
    1.0
  )
) ;orderedSpacings

```

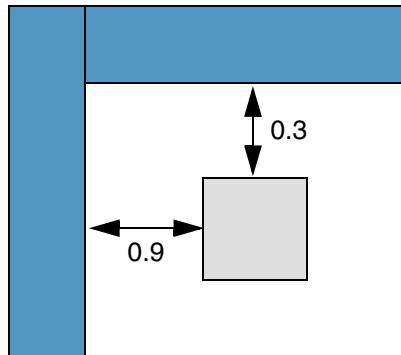
 Metal2  
 Metal1



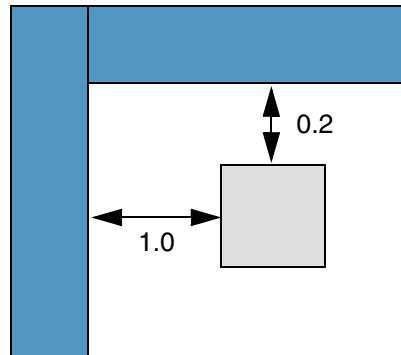
a) PASS. The top Metal2 edge satisfies the minimum spacing requirement of 0.3 and the left Metal2 edge satisfies the minimum inner vertex spacing requirement of 1.0.



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.

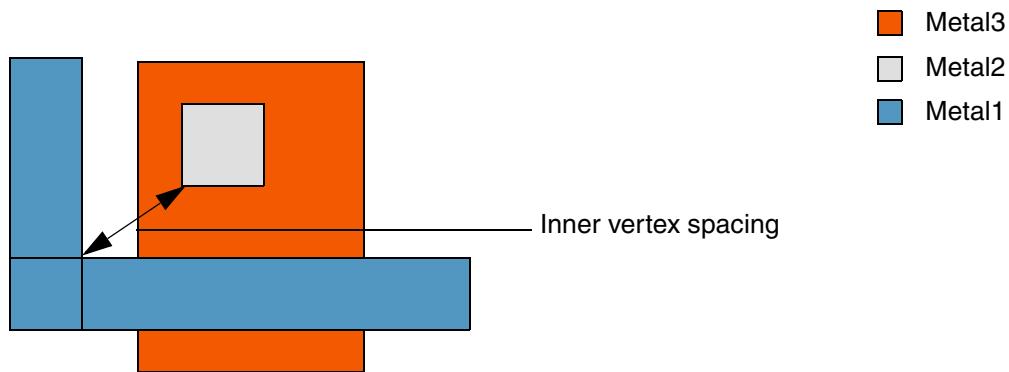


d) FAIL. The top Metal2 edge does not satisfy the minimum spacing requirement of 0.3.

## **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 Constraint 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 (Advanced Nodes Only)**

```
orderedSpacings(
  ( minNeighboringShapesSpacing tx_layer1 tx_layer2
    'widthRanges (g_widthRanges)
    'oppositeSpacingRanges (g_oppSpacingRanges)
    'lineEndSpacingRanges (g_lineEndSpacingRanges)
    ['otherLayer tx_otherLayer]
    f_spacing
  )
) ;orderedSpacings
```

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.

## 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 Constraint Reference

### Spacing Constraints (Two Layers)

---

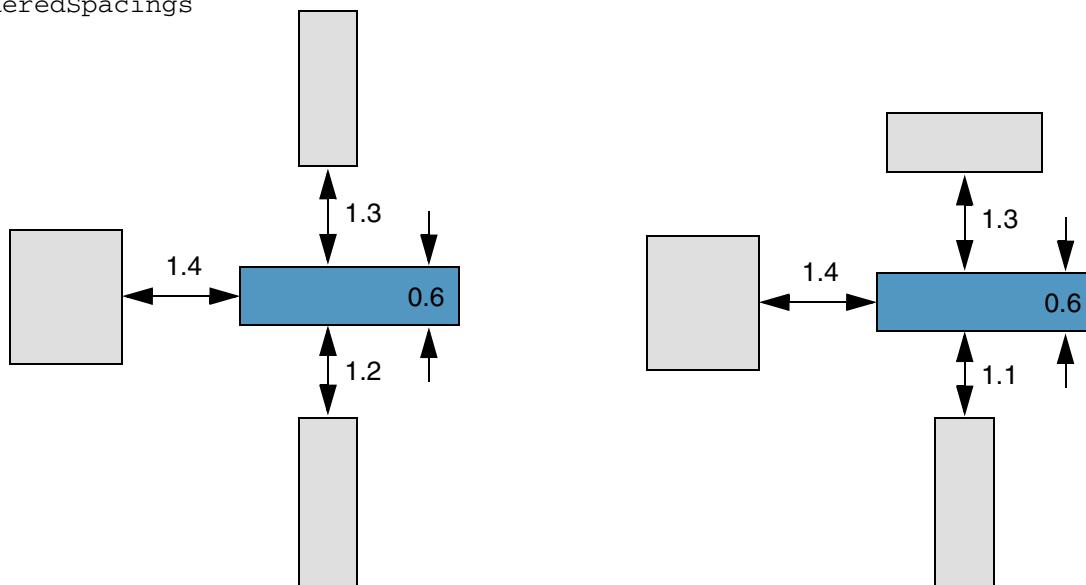
#### ***Example 1: minNeighboringShapesSpacing with widthRanges, oppositeSpacingRanges, and lineEndSpacingRanges***

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) **FAIL.** 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 Constraint Reference

### Spacing Constraints (Two Layers)

---

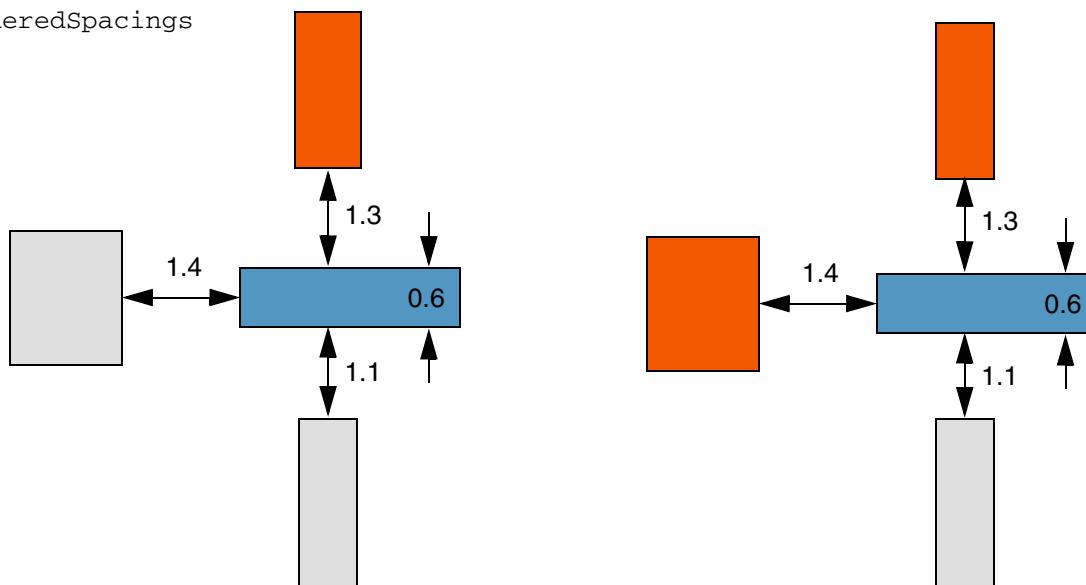
#### ***Example 2: minNeighboringShapesSpacing with widthRanges, oppositeSpacingRanges, lineEndSpacingRanges, and otherLayer***

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
```

<span style="color: orange;">█</span>	Metal3
<span style="background-color: #e0e0e0; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span>	Metal2
<span style="background-color: #0070C0; border: 1px solid black; display: inline-block; width: 10px; height: 10px;"></span>	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.

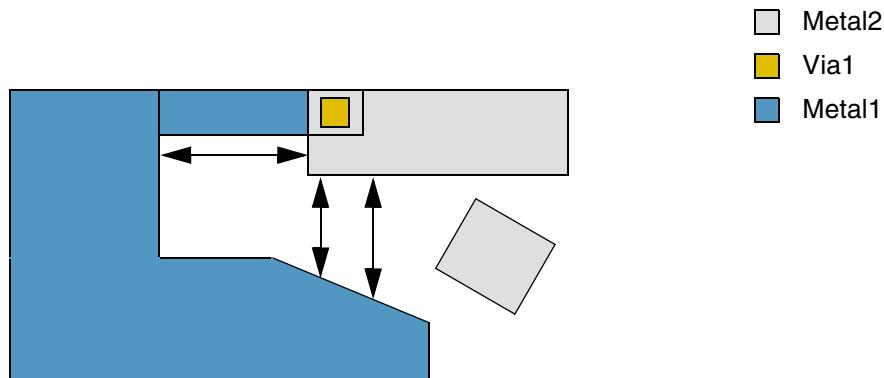
b) FAIL. The constraint applies because the width of the Metal1 shape is in the range [0.6 0.7]; the short edge of the top Metal3 shape faces the Metal2 shape on the opposite side and its distance from the Metal1 shape is 1.3 (in the range [1.0 1.4]); and the distance of the Metal2 shape on the left 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).

## 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 Constraint 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) (Advanced Nodes Only)

```
orderedSpacings(
    ( minSideSpacing tx_layer1 tx_layer2
        [ 'shortSideToShortSide | 'shortSideToLongSide | 'shortSideToAnySide
            | 'longSideToLongSide | 'longSideToAnySide
        ]
        [ 'prl f_prl]
        [ 'widthRanges (g_ranges)]
        [ 'otherWidthRanges (g_otherRanges)]
        [ 'horizontal | 'vertical]
        [ 'exceptOverlap f_exceptOverlap]
        [ 'cornerEuclidian]
        [ 'mask1 | 'mask2 | 'mask3]
        [ 'otherMask1 | 'otherMask2 | 'otherMask3]
        [ 'insideLayers (tx_layer3 tx_layer4 ... tx_layerN)
            | 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
        ]
        f_spacing
    )
)
; orderedSpacings
```

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)
<i>f_spacing</i>	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.

Type: Boolean

'prl *f\_prl*

The constraint applies only if the parallel run length between the two sides is greater than this value.

Type: Boolean

'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 Constraint 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*)

(ICADV12.3 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*.

'cornerEuclidian

(ICADV12.3 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

(ICADV12.3 Only) The constraint applies to this mask on *layer1*.

'otherMask1 | 'otherMask2 | 'otherMask3

(ICADV12.3 Only) The constraint applies to this mask on *layer2*.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

---

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

(ICADV12.3 Only) 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 (*tx\_layer3 tx\_layer4 ... tx\_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 (*tx\_layer3 tx\_layer4 ... tx\_layerN*).

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

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

## Examples

- [Example 1: minSideSpacing with shortSideToShortSide, prl, widthRanges, and otherWidthRanges](#)
- [Example 2a: minSideSpacing with exceptOverlap](#)
- [Example 2b: minSideSpacing with exceptOverlap](#)

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

---

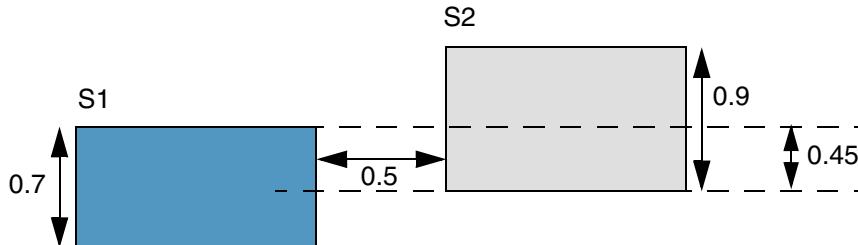
#### ***Example 1: minSideSpacing with shortSideToShortSide, prl, widthRanges, and otherWidthRanges***

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:

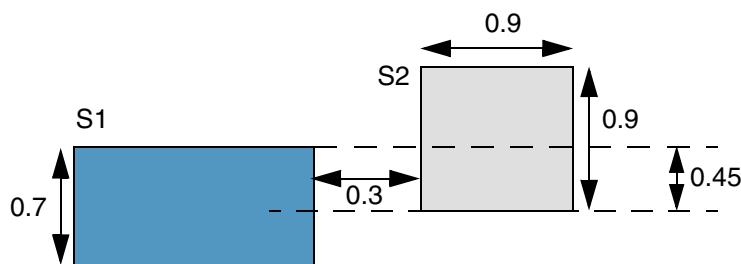
- The width of S1 is greater than or equal to 0.6 and less than or equal to 0.8.
- The width of S2 is greater than or equal to 0.9 and less than or equal to 1.0.
- The parallel run length between the sides is greater than 0.

```
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 Constraint Reference

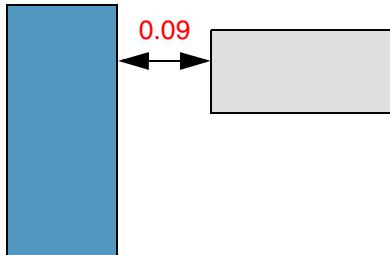
### Spacing Constraints (Two Layers)

#### Example 2a: *minSideSpacing* with *exceptOverlap*

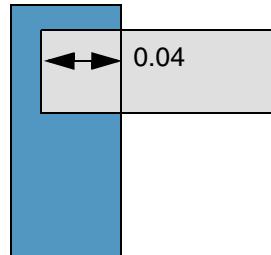
The spacing between Trim1 and Metal2 shapes must be at least 0.1, except when the horizontal overlap between the two shapes is greater than or equal to 0.05.

```
orderedSpacings(  
    ( minSideSpacing "Trim1" "Metal2"  
        'horizontal  
        'exceptOverlap 0.05  
        0.1  
    )  
) ;orderedSpacings
```

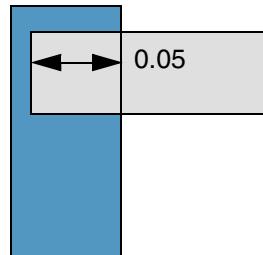
 Metal2  
 Trim1



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 Constraint Reference

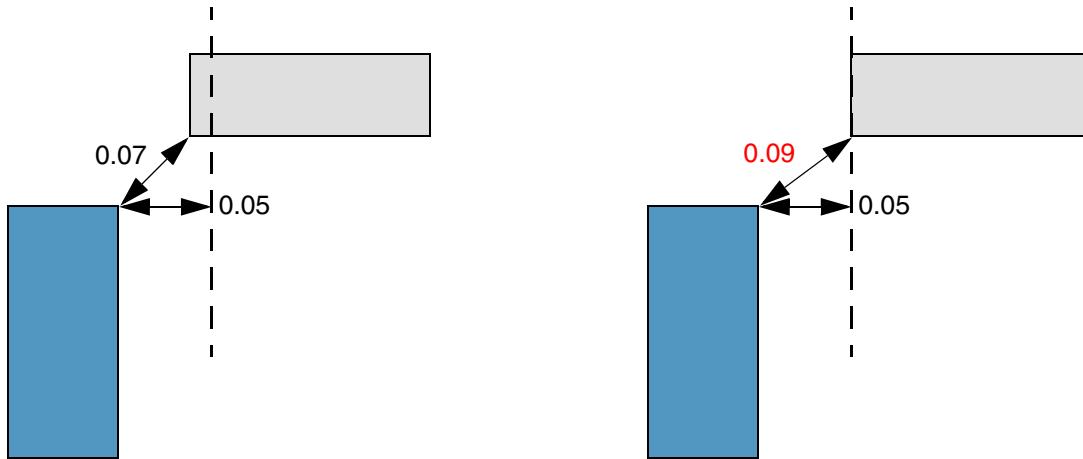
### Spacing Constraints (Two Layers)

#### 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 exactly equal to 0.05. Therefore, the spacing requirement of 0.1 must be satisfied.

## minSpacing (Two layers)

```
spacings(
  ( minSpacing tx_layer1 tx_layer2
    ['orthogonalSpacing f_orthoSpacing]
    ['horizontal | 'vertical]
    ['overlapNotAllowed]
    ['crossingAllowed]
    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
```

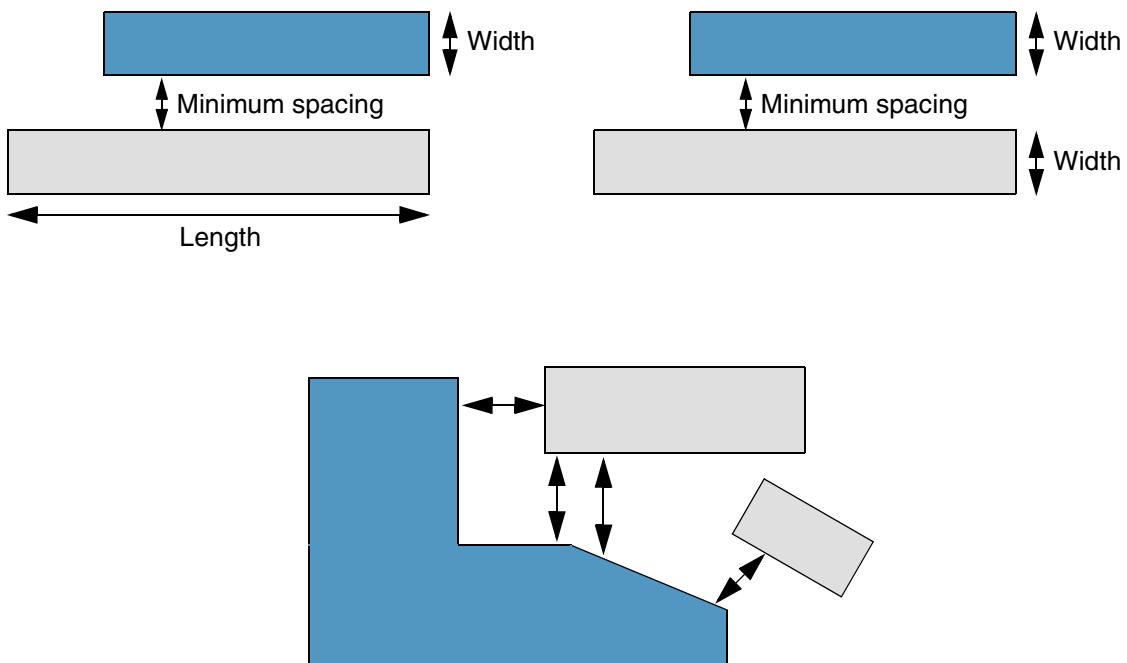
Specifies the minimum edge-to-edge spacing required between two adjacent, non-intersecting shapes on two different layers. The required spacing can be dependent on the direction in which the spacing is measured, on the width of both the shapes or on the width of the shapes and the parallel run length between them.

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

The figures below illustrate how the constraint is applied:

Legend:  
Metal2 (Light Gray)  
Metal1 (Dark Blue)

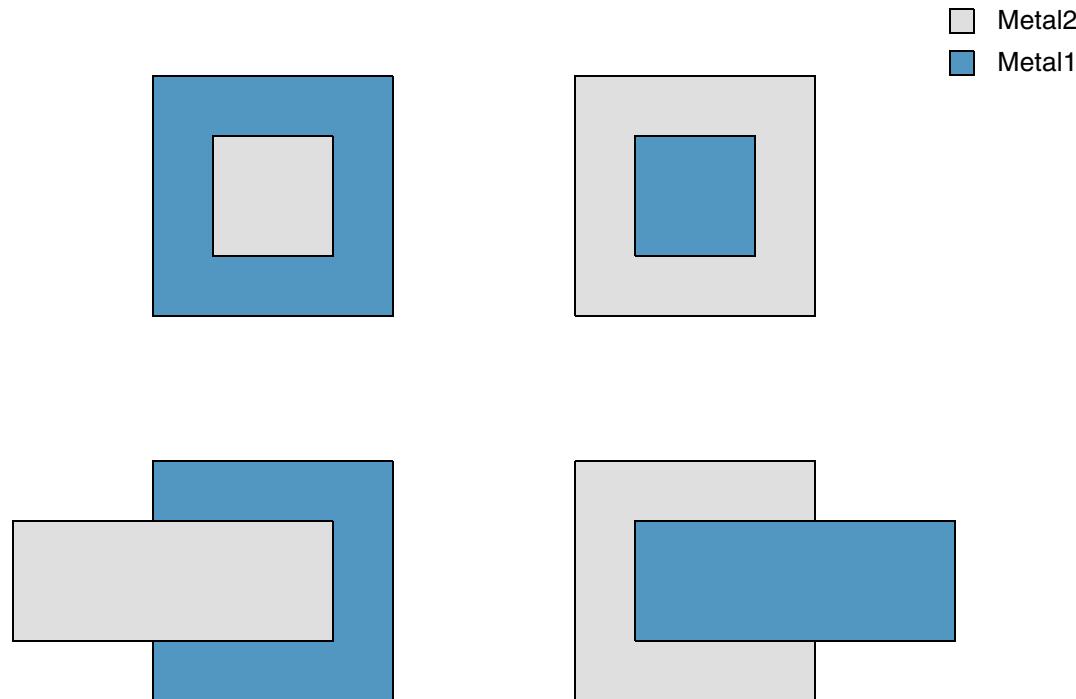


All distances indicated by arrows must be greater than or equal to the specified minimum spacing value.

## Virtuoso Technology Data Constraint 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.

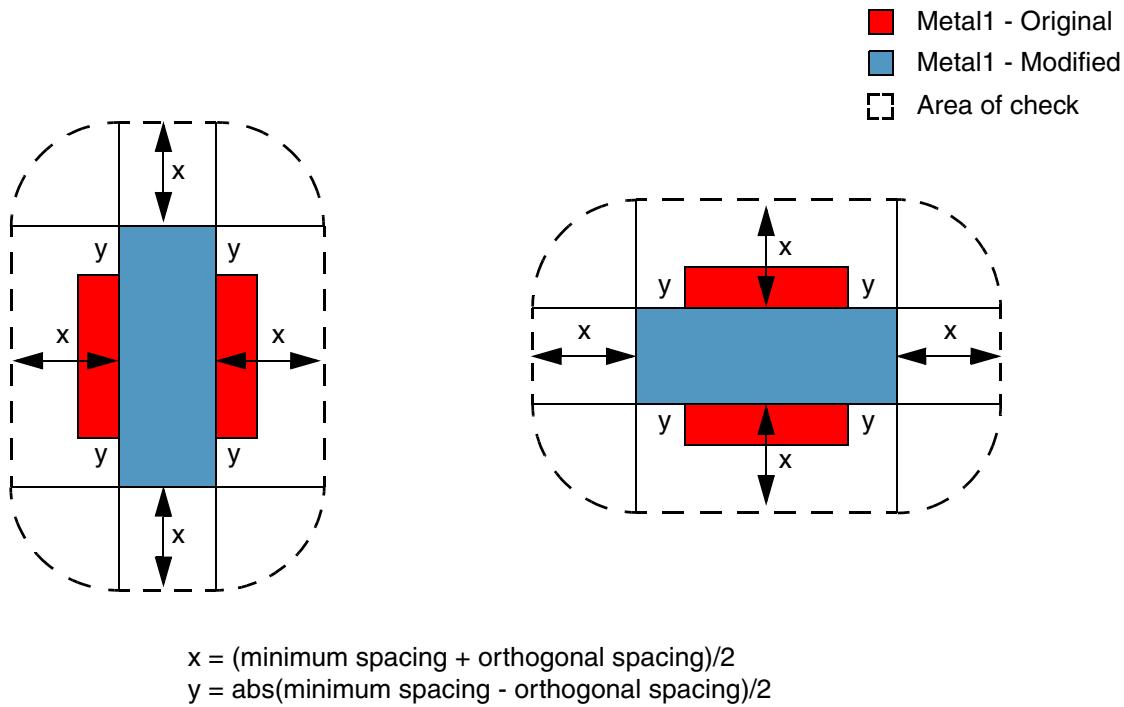
The minimum orthogonal spacing check is determined by the following method: A pair of opposite sides of the `layer1` shape is grown by a distance equal to  $\text{abs}(\text{minimum spacing} - \text{orthogonal spacing})/2$ , denoted by "y" in the figure below. Similarly, the other pair of opposite sides 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 Constraint 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.



### Values

*tx\_layer1*

The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

*f\_spacing*

The minimum spacing required between two shapes.

"width" nil nil "width" nil nil

This identifies the index for *table*.

## Virtuoso Technology Data Constraint 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 indices.

**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 indices (and not greater than or equal to).

Type: A 2-D table specifying floating-point width, parallel run length, and spacing values.

## Virtuoso Technology Data Constraint 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

(Advanced Nodes Only) Shapes are not allowed to overlap. Otherwise, same-net shapes can overlap.

'crossingAllowed

(ICADV12.3 Only) The constraint does not apply to crossing shapes.

'manhattan

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

By default, the constraint uses Euclidean measurement.

Type: Boolean

'coincidentAllowed

The edges the two shapes can coincide. Otherwise, the shapes must meet the spacing requirement.

Type: Boolean

#### Examples

- [Example 1: minSpacing with vertical](#)
- [Example 2: minSpacing with orthogonalSpacing](#)
- [Example 3: minSpacing with crossingAllowed](#)

## Virtuoso Technology Data Constraint Reference

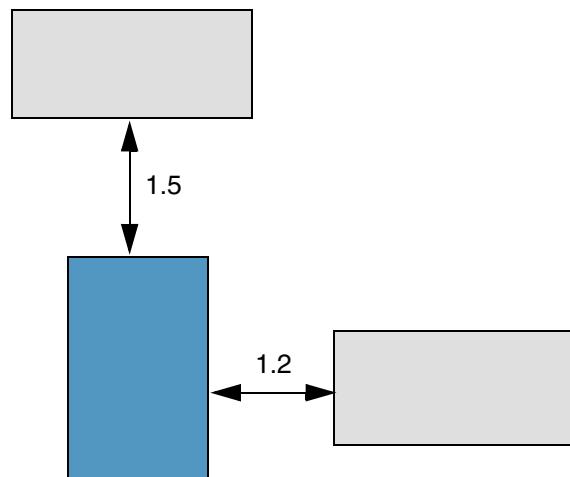
### Spacing Constraints (Two Layers)

#### ***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 Constraint Reference

### Spacing Constraints (Two Layers)

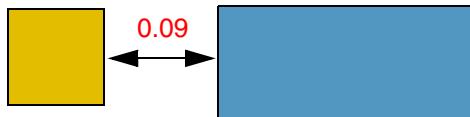
---

#### **Example 2: minSpacing with orthogonalSpacing**

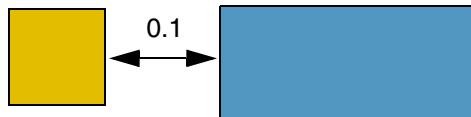
The spacing between Via1 and Metal1 shapes must be at least 0.1. Orthogonal spacing specified is 0.2, which means that Via1 shapes are grown and shrunk by 0.05 ( $(0.2 - 0.1)/2 = 0.05$ ) and the spacing required in this case is 0.15 ( $(0.2 + 0.1)/2 = 0.15$ ).

```
spacings(
    ( minSpacing "Via1" "Metal1"
        'orthogonalSpacing 0.2
        0.1
    )
) ;spacings
```

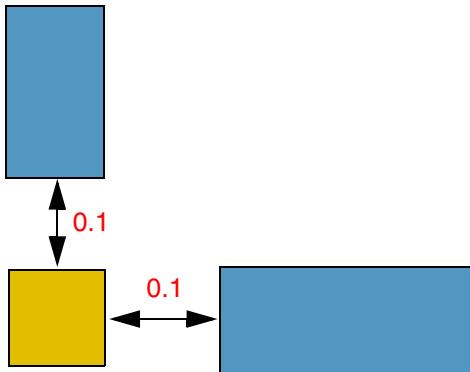
█ Via1  
█ Metal1



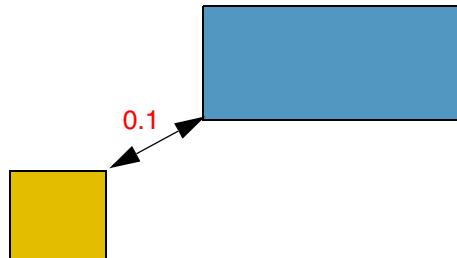
a) FAIL. The spacings between the two shapes after shrinking and growing the Via1 shape are 0.14 and 0.04, respectively, which are both less than the required spacing of 0.15.



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 Constraint Reference

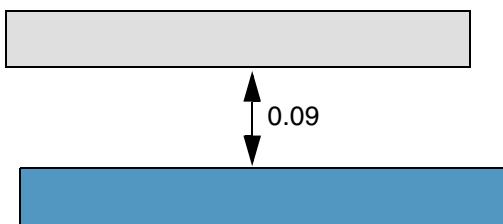
### Spacing Constraints (Two Layers)

#### **Example 3: minSpacing with crossingAllowed**

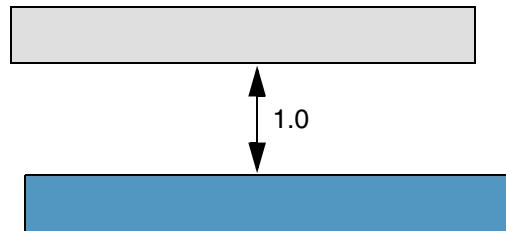
The spacing between Metal1 and Metal2 shapes must be at least 0.1. The constraint does not apply to crossing wires.

```
spacings(
  ( minSpacing "Metal1" "Metal2"
    'crossingAllowed
    0.1
  )
) ;spacings
```

 Metal2  
 Metal1



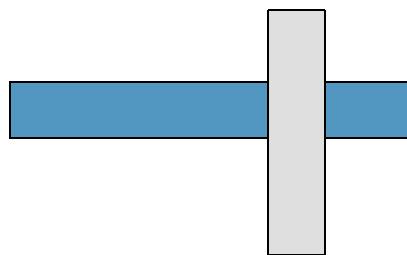
a) FAIL. The spacing between the two shapes is only 0.09 (<0.1).



b) PASS. The spacing between the two shapes is 1.0, which satisfies the constraint.



c) FAIL. Parallel shapes cannot touch.



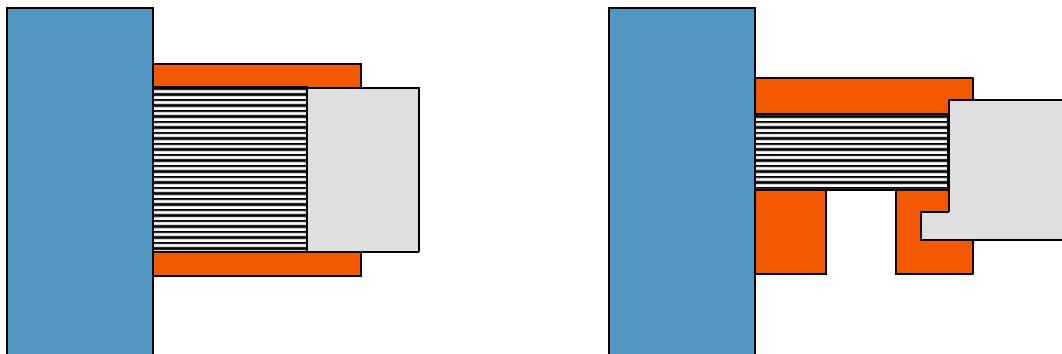
d) The constraint does not apply because 'crossingAllowed' is specified.

## minSpacingOver

```
orderedSpacings(  
  ( minSpacingOver tx_layer1 tx_layer2 tx_layer3  
    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)

*tx\_layer2*

The second layer on which the constraint is applied.

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

## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

---

<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

None

## 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.

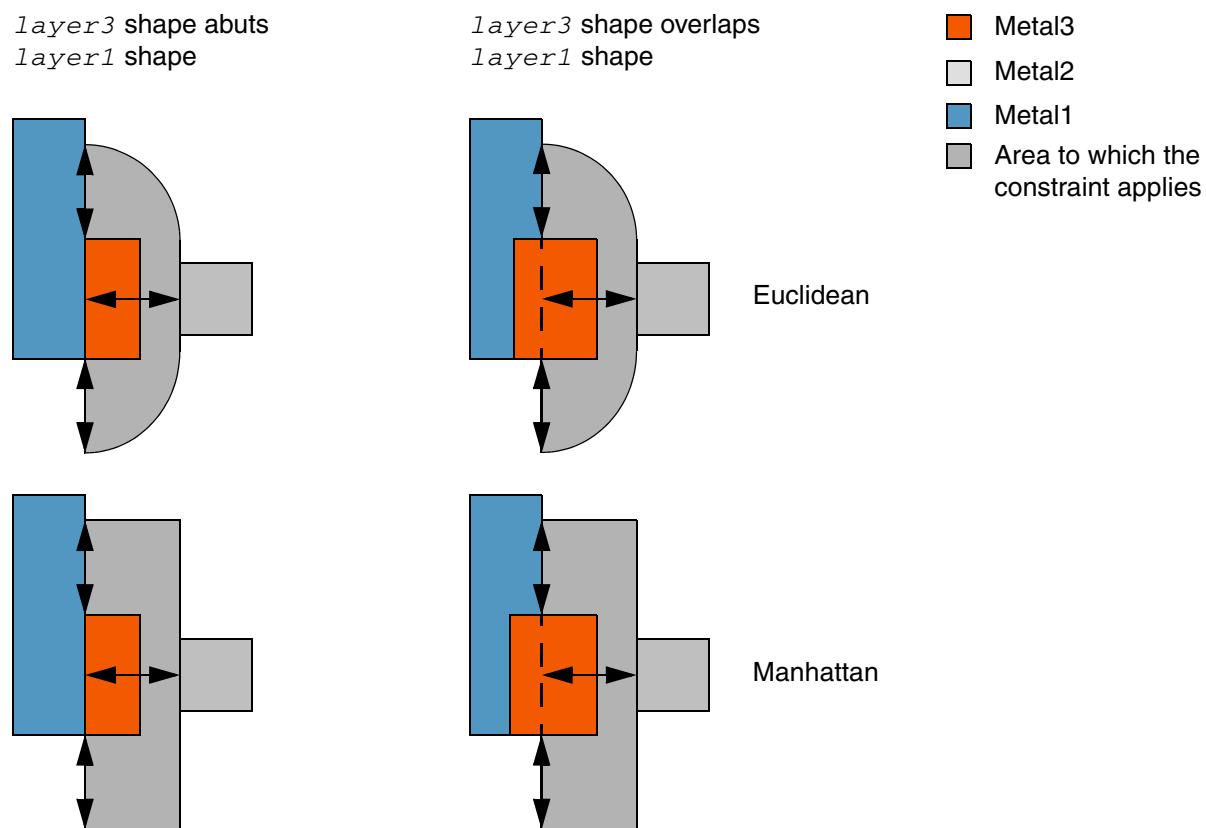
```
orderedSpacings(  
  ( minSpacingOver "Metal1" "Metal2" "Poly"  
    2.0  
  )  
) ;orderedSpacings
```

## 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.



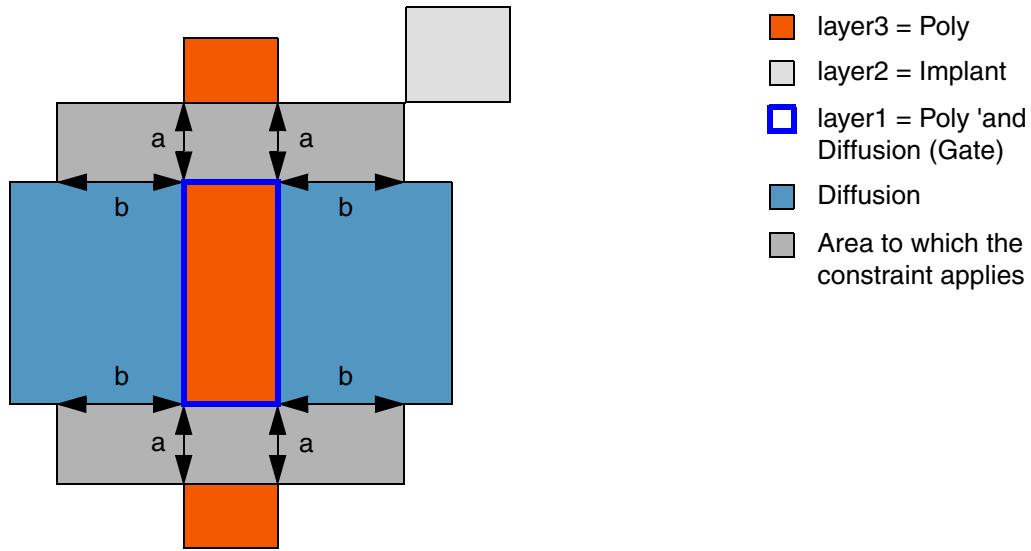
If both *spacing* and *oppSpacing* are specified, *spacing* describes the minimum spacing in the direction perpendicular to the *layer1* edge abutted or overlapped by the *layer3* shape and *oppSpacing* describes the minimum spacing parallel to this *layer1* edge. The distance must be measured as Manhattan, as shown below.

## Virtuoso Technology Data Constraint Reference

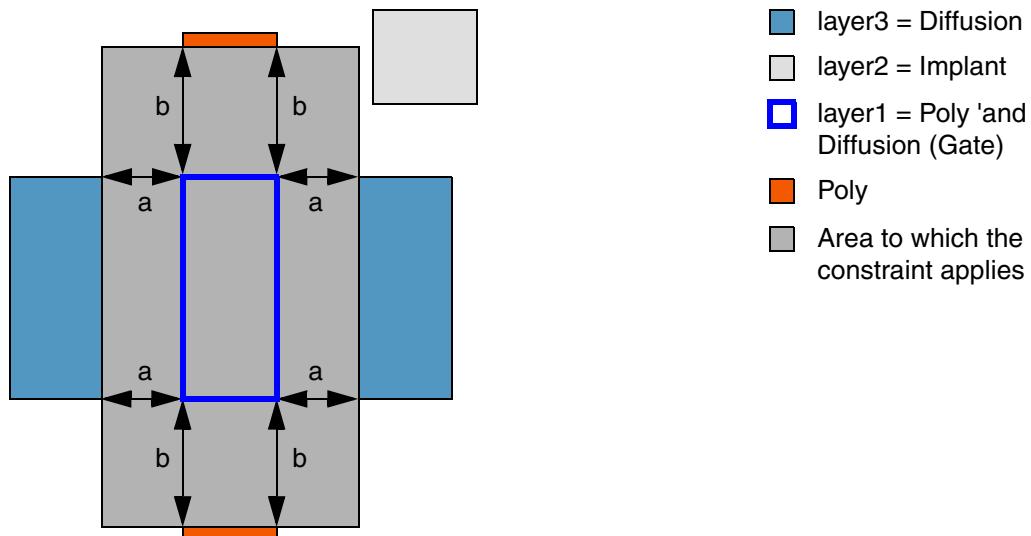
### Spacing Constraints (Two Layers)

---

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.



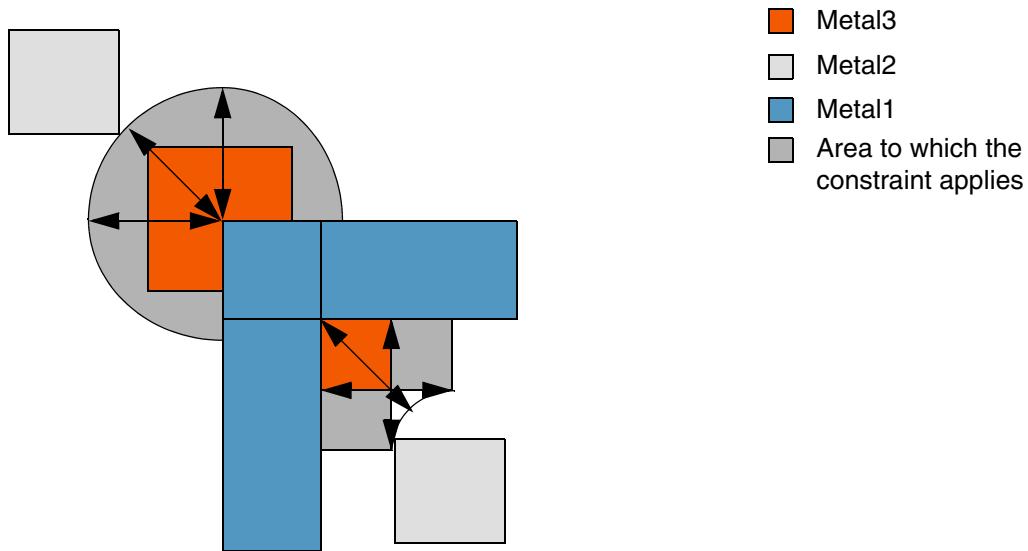
In this second example, in which `layer3` is set to Diffusion, the spacing to the Implant layer must be at least  $a$  (`spacing`) in the direction of the touching Diffusion layer (that is, in the direction perpendicular to the `layer1` edge overlapped by the `layer3` shape) and  $b$  (`oppSpacing`) in the direction parallel to this `layer1` edge.



## Virtuoso Technology Data Constraint Reference

### Spacing Constraints (Two Layers)

The constraint also applies to corners where *layer3* and *layer1* touch, as shown in the figure below.



## Values

*tx\_layer1*

The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

*tx\_layer3*

The reference layer. The placement of a *layer3* shape between *layer1* and *layer2* shapes determines how the constraint is applied.

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

*f\_spacing*

The minimum spacing in the direction perpendicular to the *layer1* edge abutted or overlapped by the *layer3* shape.

*f\_oppSpacing*

The minimum spacing in the direction parallel to the *layer1* edge abutted or overlapped by the *layer3* shape.

## Virtuoso Technology Data Constraint 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.

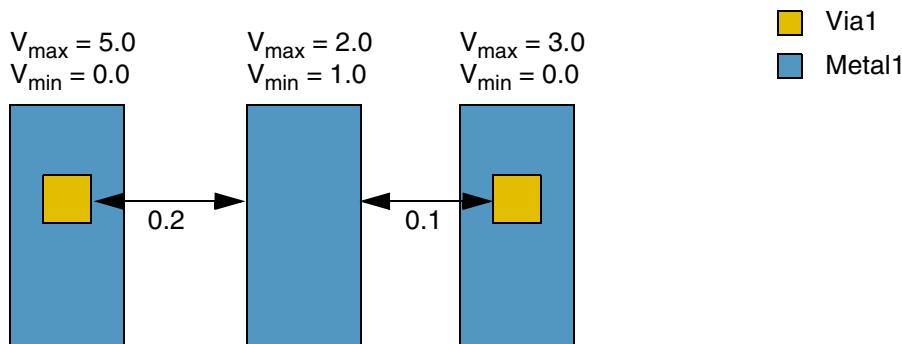
Type: Boolean

## 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)

"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 actual 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

(ICADV12.3 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 Constraint 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 "Metall1" "Vial1"
    (( "voltage" nil nil ))
    (
      0.0  0.065
      1.5  0.085
      3.3  0.105
    )
  ) ;spacingTables
```

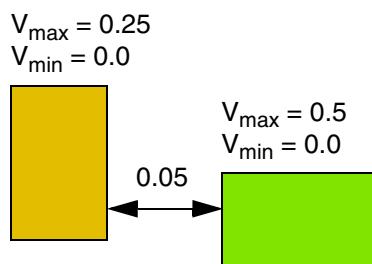
**Example 2: minVoltageSpacing with two cut layers and spacing direction**

The horizontal spacing between a Via1 cut shape and a Via2 cut shape must be at least:

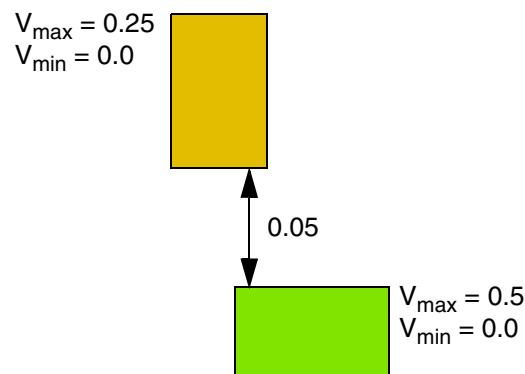
- 0.076 if the maximum voltage difference between the cut shapes is greater than or equal to 0.0V and less than 1.0V
- 0.105 if the maximum voltage difference between the cut shapes is greater than or equal to 1.0V

```
spacingTables(
  ( minVoltageSpacing "Via1" "Via2"
    ( ( "voltage" nil nil )
      'horizontal
    )
    (
      0.0  0.076
      1.0  0.105
    )
  ) ;spacingTables
```

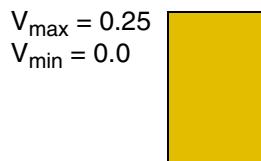
 Via2  
 Via1



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 be 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

*tx\_layer1*

The first layer on which the constraint is applied.

Type: String (layer and purpose names) 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)

*g\_ranges*

The allowed spacing ranges between shapes on different layers.

Type: Floating-point values specifying allowed spacing ranges.

"width" nil nil

This identifies the index for *table*.

*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 Constraint Reference

### Spacing Constraints (Two Layers)

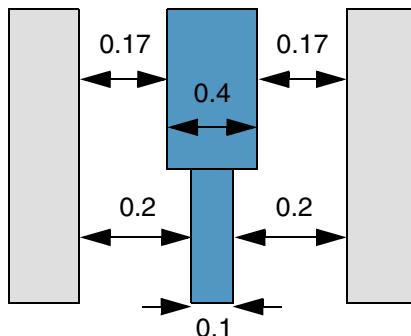
---

#### ***Example 1: shapeRequiredSpacing with horizontal, twoSides, and widthRanges***

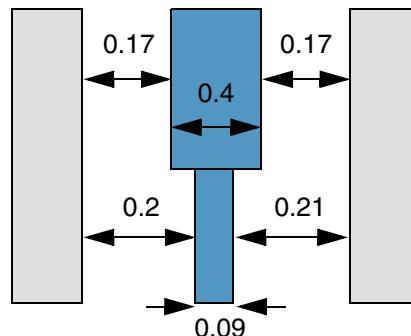
The horizontal spacing between a Metal1 shape and a Metal2 shape must be greater than or equal to 0.1 and less than or equal to 0.2 if Metal2 shapes are present along its full parallel run length on both sides, provided the width of the Metal1 shape is greater than or equal to 0.1 and less than or equal to 0.4.

```
orderedSpacings(
  ( shapeRequiredSpacing "Metal1" "Metal2"
    'horizontal 'twoSides
    'widthRanges ("[0.1 0.4]")
    ("[0.1 0.2]")
  )
) ;orderedSpacings
```

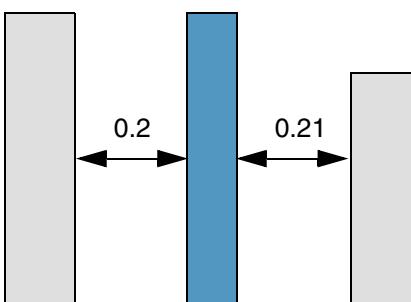
 Metal2  
 Metal1



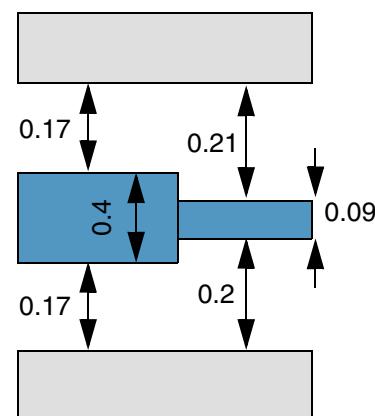
a) **PASS.** The width of the Metal1 shape is in the range [0.1 0.4] and Metal2 shapes are present along its full parallel run length on both sides. Additionally, the spacing between the Metal1 and Metal2 shapes is in the range [0.1 0.2].



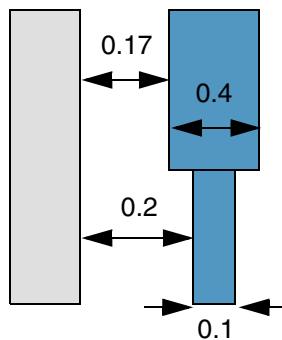
b) **FAIL.** The spacing between the lower portion of the Metal1 shape and the Metal2 shape is 0.21, which is outside the allowed range of [0.1 0.2].



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.



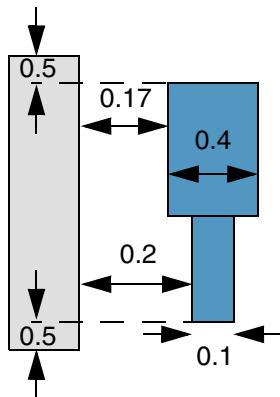
e) FAIL. A Metal2 shape is present on only one side of the Metal1 shape.

**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].

## **trimMinSpacing (Two layers) (ICADV12.3 Only)**

```
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)
      | 'outsideLayers (tx_layer3 tx_layer4 ... tx_layerN)
    ]
    f_spacing
  )
) ;spacings
```

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 Constraint 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 Constraint 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 (*tx\_layer3 tx\_layer4 ... tx\_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 (*tx\_layer3 tx\_layer4 ... tx\_layerN*).

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

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

---

## Via Construction Constraints

---

This chapter includes the following constraints:

- [allowedCutClass \(Advanced Nodes Only\)](#)
- [definedCutClassesOnly](#)
- [forbiddenCutClassSpacingRange \(Advanced Nodes Only\)](#)
- [largeRectViaArrayAllowed](#)
- [minCutClassSpacing \(One layer\)](#)
- [minCutClassSpacing \(Two layers\)](#)
- [minEndOfLineCutSpacing](#)
- [minLargeViaArrayCutSpacing](#)
- [minLargeViaArraySpacing](#)
- [minLargeViaArrayWidth](#)
- [minNeighborViaSpacing](#)
- [minOrthogonalViaSpacing](#)
- [minParallelViaSpacing \(One layer\)](#)
- [minParallelViaSpacing \(Two layers plus metal\)](#)
- [minParallelWithinViaSpacing](#)
- [minSameMetalSharedEdgeViaSpacing](#)
- [minViaSpacing \(One layer\)](#)
- [minViaSpacing \(Two layers\)](#)
- [redundantViaSetback](#)
- [stackable](#)

## **Virtuoso Technology Data Constraint Reference**

### Via Construction Constraints

---

- [viaEdgeType](#)
- [viaRequiredWithin \(Advanced Nodes Only\)](#)
- [viaSpacing](#)
- [viaStackingLimits](#)

## allowedCutClass (Advanced Nodes Only)

```
spacingTables(
    ( allowedCutClass tx_cutLayer
        (( "width" nil nil "width" nil nil )
         'cutClass {f_width | (f_width f_length) | t_name}
         ['lowerLayer tx_lowerLayer]
         ['upperLayer tx_upperLayer]
         ['horizontalLowerDir | 'verticalLowerDir]
         ['horizontalUpperDir | 'verticalUpperDir]
         x_default
     )
     (g_table)
   )
) ;spacingTables
```

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.

### 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 the specified cut class is 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

'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

'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*

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

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints

---

*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.

### Examples

- [Example 1: allowedCutClass with cutClass, lowerLayer, and upperLayer](#)
- [Example 2: allowedCutClass with cutClass, lowerLayer, upperLayer, horizontalLowerDir, and horizontalUpperDir](#)

**Virtuoso Technology Data Constraint Reference**  
Via Construction Constraints

---

**Example 1: allowedCutClass with cutClass, lowerLayer, and upperLayer**

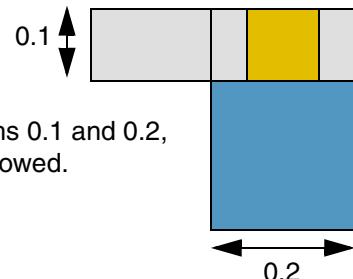
- The V1\_small cut class is not allowed between Metal1 and Metal2 wires if one or both wires are greater than or equal to 0.1 wide.
- The V1\_large cut class is not allowed between Metal1 and Metal2 wires if one or both wires are 0.1 wide or if both wires are greater than or equal to 0.2 wide.

```

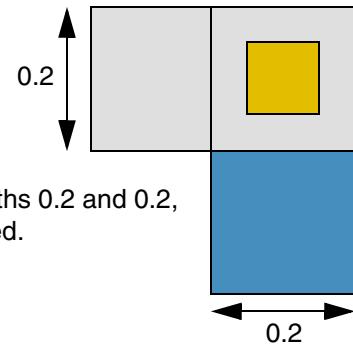
spacingTables(
    ( allowedCutClass "Via1"
        (( "width" nil nil "width" nil nil )
         'cutClass "V1_small"
         'lowerLayer "Metal1"
         'upperLayer "Metal2"
         1
     )
    (
        (0.1 0.1) 0
        (0.1 0.2) 0
        (0.1 0.3) 0
        (0.1 0.4) 0
        (0.2 0.1) 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

```

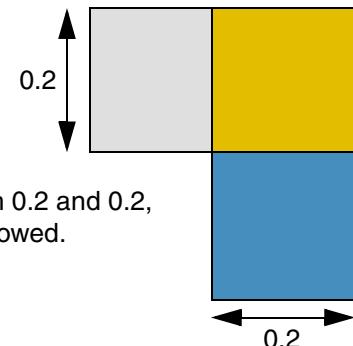
  
 Metal2  
 Via1  
 Metal1



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.

***Example 2: 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
```

## **definedCutClassesOnly**

```
    spacings(  
        ( definedCutClassesOnly tx_layer )  
    ) ;spacings
```

Requires that all cut shapes on the specified layer must match the `cutClasses` defined for the layer.

## Values

*tx\_layer* 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 "Vial1" )  
) ;spacings
```

## **forbiddenCutClassSpacingRange (Advanced Nodes Only)**

```
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]
    ['cutClassPrl g_cutClassPrl | 'allCuts]
    (g_range)
  )
) ;spacings
```

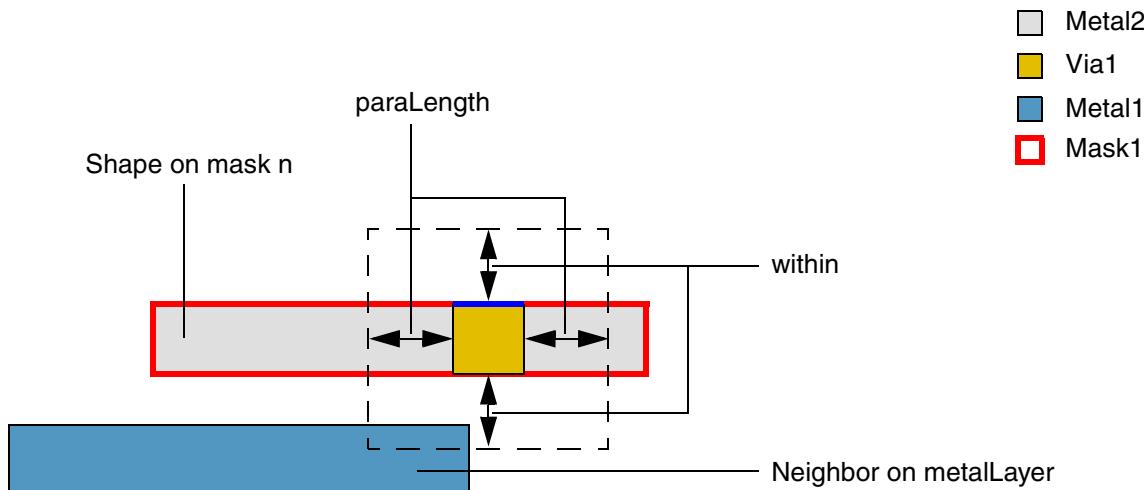
Specifies that the spacing between two cut shapes of the specified cut class on the specified layer cannot fall in the specified range.

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.

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* <= spacing <= *f\_max*

Type: Floating-point values specifying a range of spacings that are not allowed.

## 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*

(ICADV12.3 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

(ICADV12.3 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*

(ICADV12.3 Only) The constraint applies only if the width of the metal neighbor is less than or equal to this value.

'paraLength *f\_paraLength*

(ICADV12.3 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*

(ICADV12.3 Only) The constraint applies only if the distance between the cut shape and the metal neighbor is less than this value.

'sameMask

(ICADV12.3 Only) The constraint applies only between same-mask cut shapes of the specified cut class.

Type: Boolean

'shortEdge | 'longEdge

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: (ICADV12.3 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.

Type: Boolean

'cutClassPrl *g\_cutClassPrl*

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\_prl ...* )

where,

- *f\_width* is the width of the cut class.
- *f\_length* is the length of the cut class.
- *f\_prl* is the parallel run length to a cut shape with these dimensions.

Type: An array specifying floating-point values.

'allCuts

(ICADV12.3 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

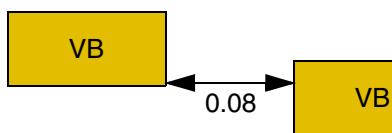
- [Example 1: forbiddenCutClassSpacingRange with shortEdge](#)
- [Example 2: forbiddenCutClassSpacingRange with cutClassPrl](#)

### ***Example 1: forbiddenCutClassSpacingRange with shortEdge***

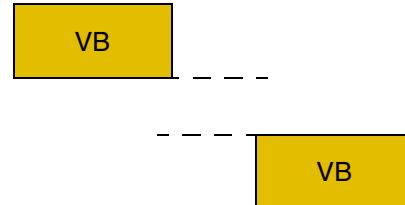
The spacing between the short edges of two VB cut shapes on layer Via1 must not be greater than or equal to 0.07 and less than or equal to 0.09 if the cut shapes have parallel run length greater than zero.

```
spacings(
  ( forbiddenCutClassSpacingRange "Via1"
    'cutClass ("VB")
    'shortEdge
    "[0.07 0.09]"
  )
) ;spacings
```

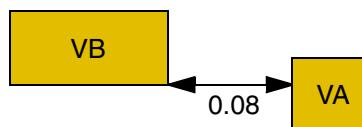
■ Via1



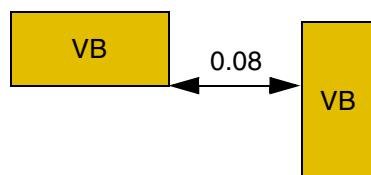
a) FAIL. The spacing between two short edges is 0.08, which lies in the forbidden spacing range of [0.07 0.09].



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.



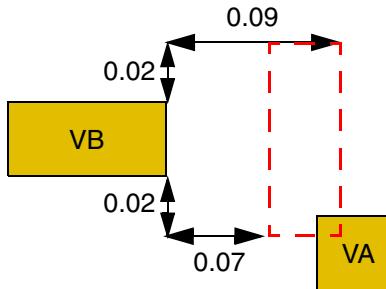
d) The constraint does not apply because there is no parallel run length between the short edges of the two cut shapes. A violation would occur if 'shortEdge' was not specified.

**Example 2: *forbiddenCutClassSpacingRange* with *cutClassPrl***

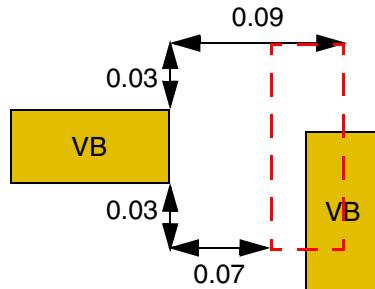
The spacing between the short edge of a VB cut shape to any edge of a VA or VB cut on layer Via1 must not be greater than or equal to 0.07 and less than or equal to 0.09. The spacing applies when the parallel run length between VA and VB cut shapes is greater than -0.02 and between two VB shapes is greater than -0.03.

```
spacings(
  ( forbiddenCutClassSpacingRange "Via1"
    'cutClass ("VB")
    'shortEdge
    'cutClassPrl (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).



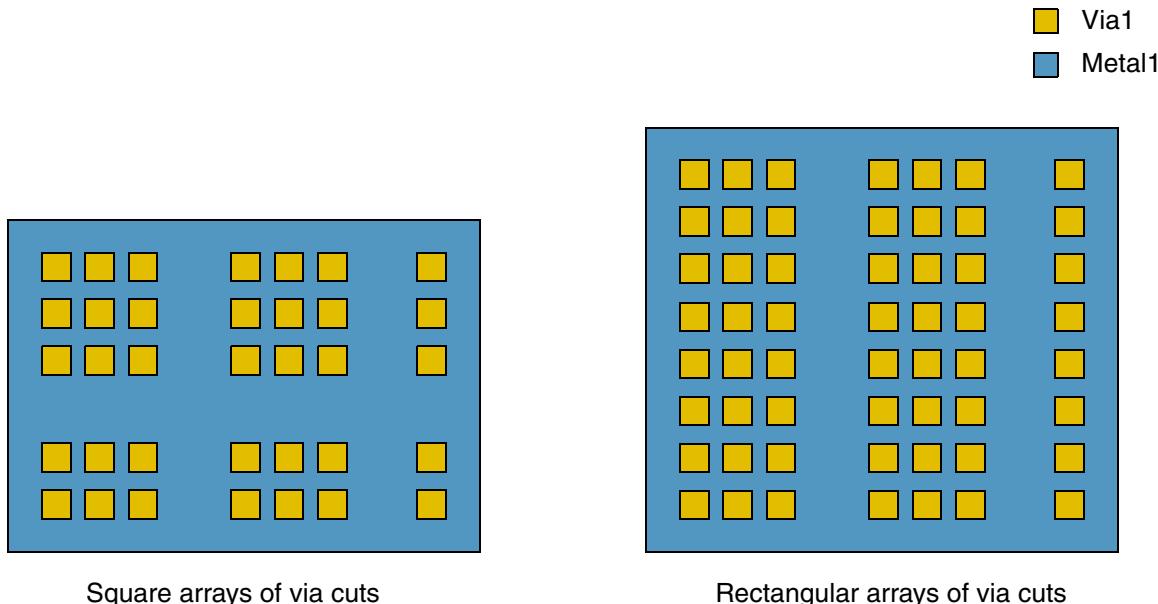
b) FAIL. The VB cut shape on the right overlaps the forbidden spacing region (indicated by the red dotted line). The constraint would not apply if '*cutClassPrl*' was not specified.

## 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.



## 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>t</i>   <i>nil</i>	If <i>t</i> , rectangular arrays are allowed; if <i>nil</i> , only square arrays are allowed.

## 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

'paraOverlap

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
```

## 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)
            )
         ]
         [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 Constraint 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 Constraint 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 Constraint 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

(Advanced Nodes Only) The constraint applies only to via cuts on the same mask.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints

---

'bothNegativePrls

(Advanced Nodes Only) 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 actual Euclidean spacing.

**Note:** If this parameter is specified for one minCutClassSpacing constraint defined on a particular layer, it must be specified for all minCutClassSpacing constraints defined on that layer.

Type: Boolean

'exactAligned ((*f\_width* | *t\_name f\_spacing*) ...)

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.

'horizontal | 'vertical

The direction in which the constraint applies. If direction is not specified, the constraint applies in any direction.

Type: Boolean

## Virtuoso Technology Data Constraint 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>
<-1.0	The constraint does not apply
$\geq -1.0$ and $<-0.05$	-1.0
$\geq -0.05$ and $<0$	-0.05
$\geq 0$ and $<0.05$	0
$\geq 0.05$ and $<1.0$	0.05
$\geq 1.0$	1.0

'horizontalOverlap | 'verticalOverlap

(ICADV12.3 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 Constraint 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)
```

(Advanced Nodes Only) 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*.

*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 bothNegativePrls](#)
- [Example 4: minCutClassSpacing with bothNegativePrls](#)

***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 Constraint 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
```

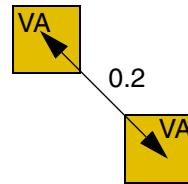
# Virtuoso Technology Data Constraint Reference

## Via Construction Constraints

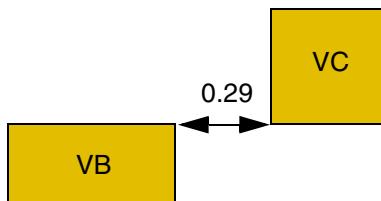
 Via1



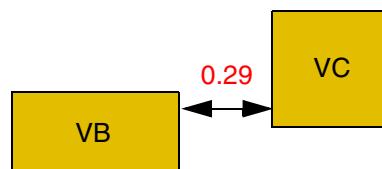
a) FAIL. The center-to-center spacing between the two shapes is 0.19 (<0.2).



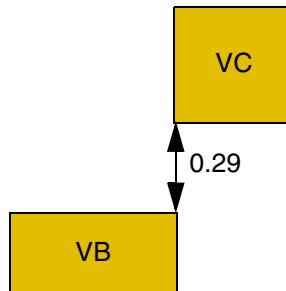
b) PASS. The center-to-center spacing between the two shapes is 0.2.



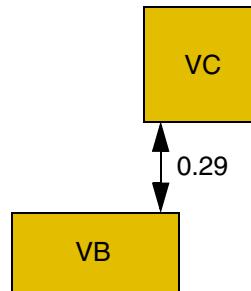
c) PASS. The parallel run length between a VB short edge and a VC edge is zero. Therefore, a default minimum spacing of 0.15 applies ( $0.29 > 0.15$ ).



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 ( $0.29 < 0.30$ ).



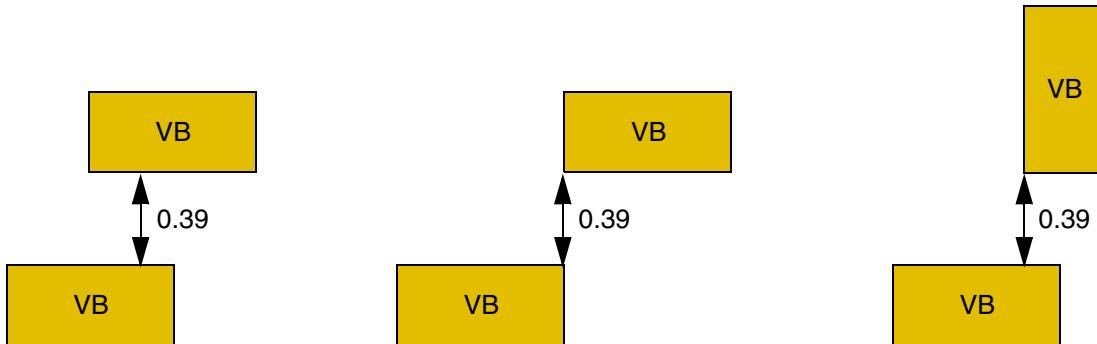
e) PASS. The parallel run length between a VB long edge and a VC edge is zero. Therefore, a default minimum spacing of 0.15 applies ( $0.29 > 0.15$ ).



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 Constraint 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 Constraint 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 Constraint 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 entries in italics are ignored. Additionally, only the dimensions of rectangular cut shapes are used to index rows.

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints

---

#### ***Example 3: minCutClassSpacing without bothNegativePrls***

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

	<b>0.1</b>	<b>0.2</b>	<b>0.2</b>	<b>0.4</b>
<b>0.1</b>	0.2	0.2	0.2	0.22
<b>0.2</b>	0.2	0.2	0.21	0.23
<b>0.2</b>	0.2	0.21	0.3	0.3
<b>0.4</b>	0.21	0.23	0.3	0.3

PRL = -0.2

	<b>0.1</b>	<b>0.2</b>	<b>0.2</b>	<b>0.4</b>
<b>0.1</b>	0.2	0.2	0.2	0.22
<b>0.2</b>	0.2	0.2	0.21	0.24
<b>0.2</b>	0.2	0.21	0.3	0.3
<b>0.4</b>	0.21	0.23	0.3	0.3

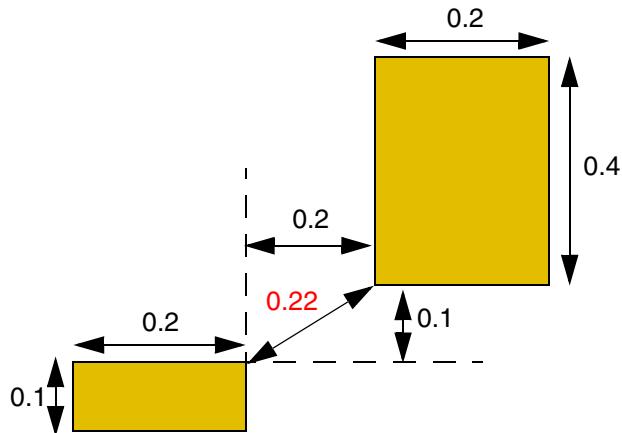
# Virtuoso Technology Data Constraint Reference

## Via Construction Constraints

```
spacingTables(                                     □ Via1
  ( 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
```

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints



FAIL. Because 'bothNegativePrls 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 actual Euclidean spacing between the shapes is 0.22, which is less than 0.23.

**Virtuoso Technology Data Constraint Reference**  
Via Construction Constraints

---

***Example 4: minCutClassSpacing with bothNegativePrls***

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

	<b>0.1</b>	<b>0.2</b>	<b>0.2</b>	<b>0.4</b>
<b>0.1</b>	0.2	0.2	0.2	0.22
<b>0.2</b>	0.2	0.2	0.21	0.23
<b>0.2</b>	0.2	0.21	0.3	0.3
<b>0.4</b>	0.21	0.23	0.3	0.3

PRL = -0.2

	<b>0.1</b>	<b>0.2</b>	<b>0.2</b>	<b>0.4</b>
<b>0.1</b>	0.2	0.2	0.2	0.22
<b>0.2</b>	0.2	0.2	0.21	0.24
<b>0.2</b>	0.2	0.21	0.3	0.3
<b>0.4</b>	0.21	0.23	0.3	0.3

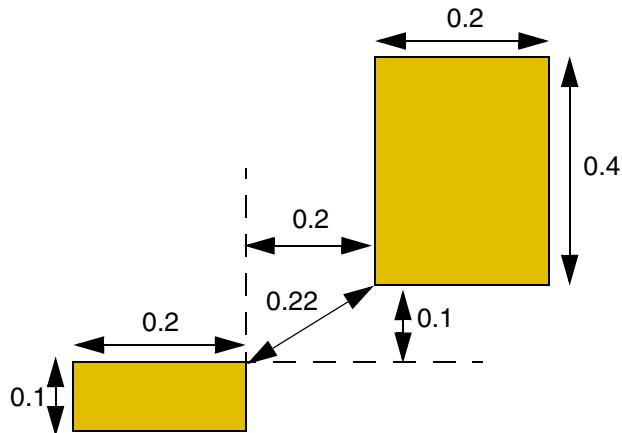
# Virtuoso Technology Data Constraint Reference

## Via Construction Constraints

```
spacingTables(                                     □ Via1
  ( 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 ) ("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
```

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints



PASS. Because 'bothNegativePrls 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 actual 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]]
     ['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 Constraint Reference

### Via Construction Constraints

---

#### Values

*tx\_cutLayer1*      The first cut layer on which the constraint is applied.  
Type: String (layer and purpose names) or Integer (layer number)

*tx\_cutLayer2*      The second 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 Constraint 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 Constraint 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

'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 Constraint 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
$\geq -1$ and $<-0.05$	-1.0
$\geq -0.05$ and $<0$	-0.05
$\geq 0$ and $<0.05$	0
$\geq 0.05$ and $<1.0$	0.05
$\geq 1.0$	1.0

'nonCutClassEdgeSpacing (*f\_length1 f\_space1*)

(Advanced Nodes Only) 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 Constraint 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
```

(Advanced Nodes Only) 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
```

(ICADV12.3 Only) The constraint applies only if the via cut to which 'nonZeroEnclosure applies has an enclosure greater than this value on all four sides.

'cutClassSizeBy ({*f\_size1 f\_size2*} ...)

(Advanced Nodes Only) 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 Constraint 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)) 0.15
        (( "VC" 'shortEdge) ("VE" 'longEdge )) 0.15
        (( "VC" 'shortEdge) ("VF" 'shortEdge)) 0.30
        (( "VC" 'shortEdge) ("VF" '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 Constraint Reference

### Via Construction Constraints

#### **Example 2: minCutClassSpacing with cutClass, cutClassProfile, sameNet, and paraOverlap**

```
spacingTables(
  ( minCutClassSpacing "Vial" "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 Constraint 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
<b>V2E</b>	SE	<b>50</b>	0.02	0.02	0.0	<i>0.0</i>	0.02	0.02
	LE	<b>100</b>	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)		<b>0.1</b>	<b>0.1</b>	<b>0.2</b>	<b>0.2</b>
W1	<b>0.1</b>	0.15	0.15	0.20	0.20
L1	<b>0.1</b>	0.15	0.15	0.20	0.20
W2	<b>0.1</b>	0.15	0.15	0.20	0.20
L2	<b>0.2</b>	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 Constraint Reference

### Via Construction Constraints

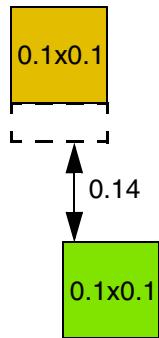
- To VA on Via2, the edges of the via cut on Via1 should be extended by 0.1.
- To VB on Via2, the edges of the via cuts on both layers should be extended by 0.01.

```
spacingTables(                                     Via2
  minCutClassSpacing "Via1" "Via2"             Via1
    (( "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
```

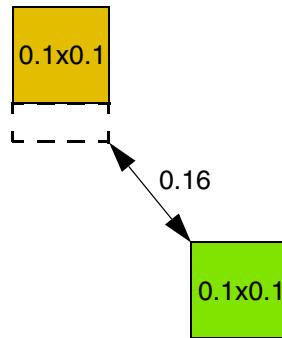
□ Sized edge

## Virtuoso Technology Data Constraint Reference

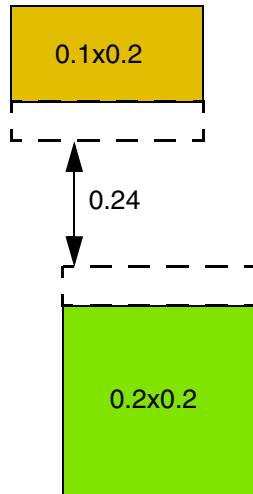
### Via Construction Constraints



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 Constraint Reference

### Via Construction Constraints

---

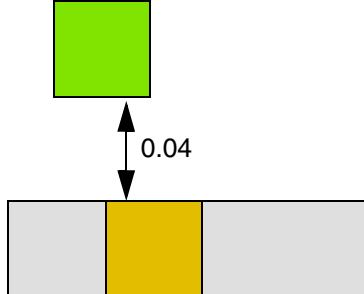
#### **Example 4: minCutClassSpacing with cutClass and nonZeroEnclosure**

The constraint applies only if the Via1 via cut has non-zero Metal2 enclosure.

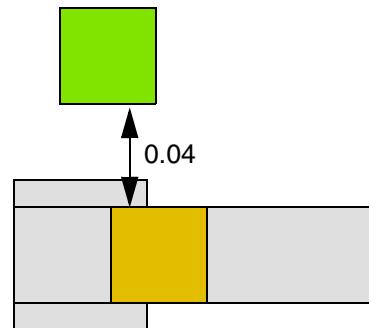
```

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

```



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).

## minEndOfLineCutSpacing

```
orderedSpacings(
    ( minEndOfLineCutSpacing tx_cutLayer tx_metalLayer
        ['cutClass { f_width | (f_width f_length) | t_name }'
         ['toCutClass
            (
                ({ f_width1 | (f_width1 f_length1) | 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 the enclosing metal shape, determines the via cut edges that must satisfy the spacing requirement. For example, in the figure below:

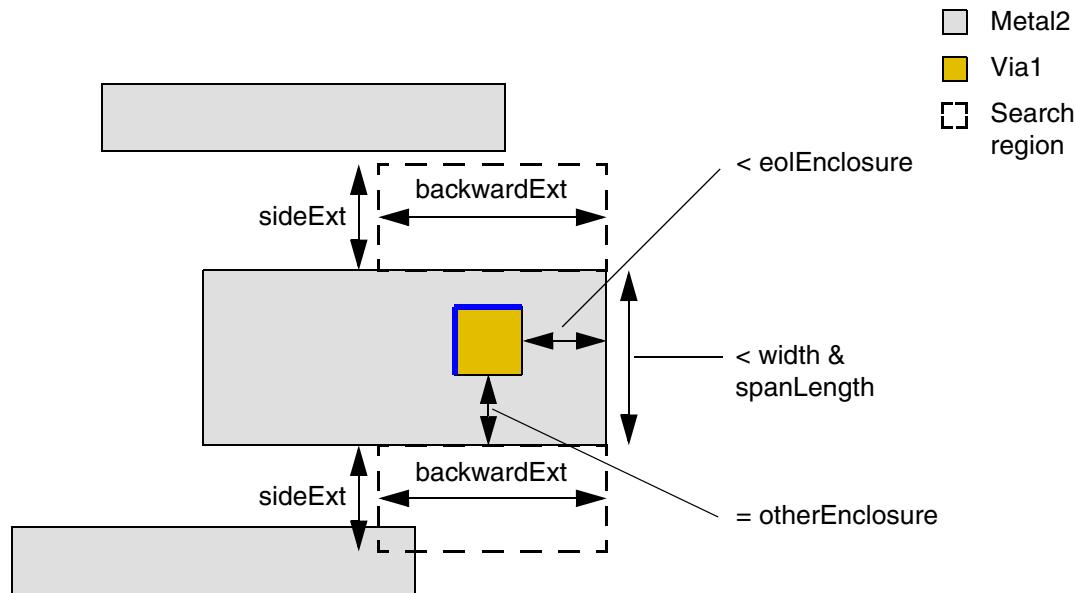
- The end-of-line width is less than *eolWidth* and the span length is less than *spanLength*.
- The end-of-line edge enclosure is less than *eolEnclosure* and the enclosure of the bottom cut edge is exactly equal to *otherEnclosure*.

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 Constraint 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:

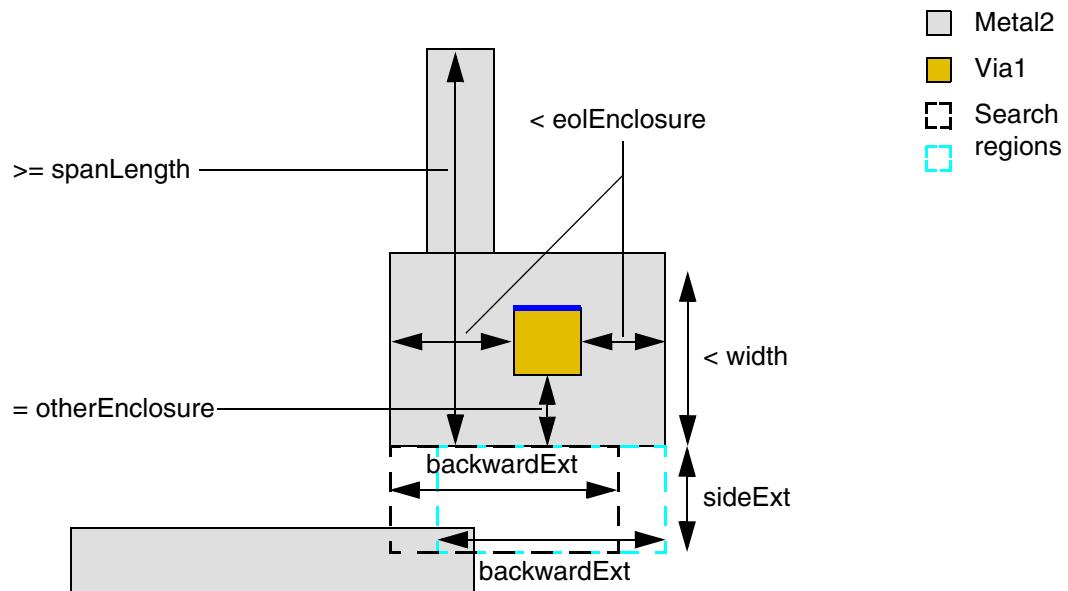
- The end-of-line width is less than *eolWidth* and the span length is greater than or equal to *spanLength*.
- The end-of-line edge enclosure is less than *eolEnclosure* and the enclosure of the bottom cut edge is exactly equal to *otherEnclosure*.

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 Constraint Reference

### Via Construction Constraints

regions—a condition that must be satisfied when span length is greater than or equal to *spanLength*.



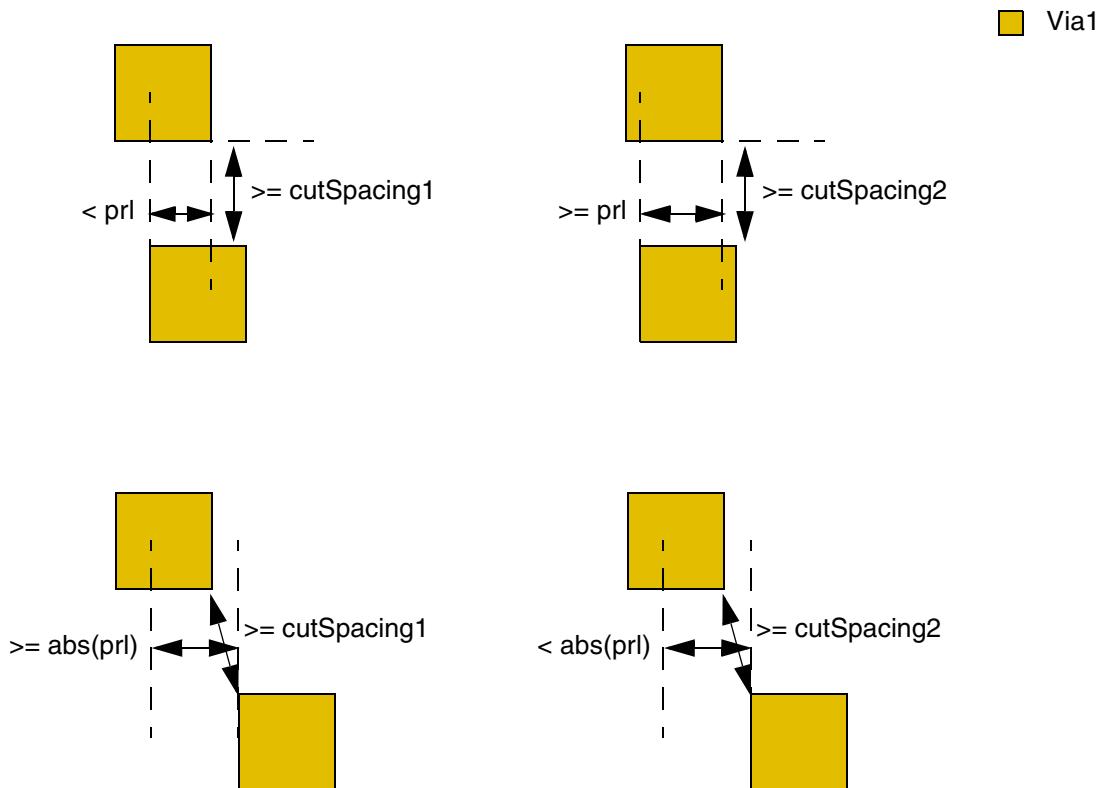
If a cut layer has cut classes, only one `minEndOfLineCutSpacing` constraint should be defined for each cut class on that cut layer. If the cut layer does not have cut classes, only one `minEndOfLineCutSpacing` constraint should be defined for that layer.

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints

---

Cut spacing based on the parallel run length between the via cuts is measured as shown below:



### Values

*tx\_cutLayer*

The cut layer on which the constraint is applied.

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

*tx\_metalLayer*

The metal layer above the cut layer on which the constraint is applied.

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

*f\_cutSpacing1* *f\_cutSpacing2*

The minimum cut spacing required based on the parallel run length (*prl*) between the two via cuts (*cutSpacing1* is typically less than *cutSpacing2*).

## 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.

'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:

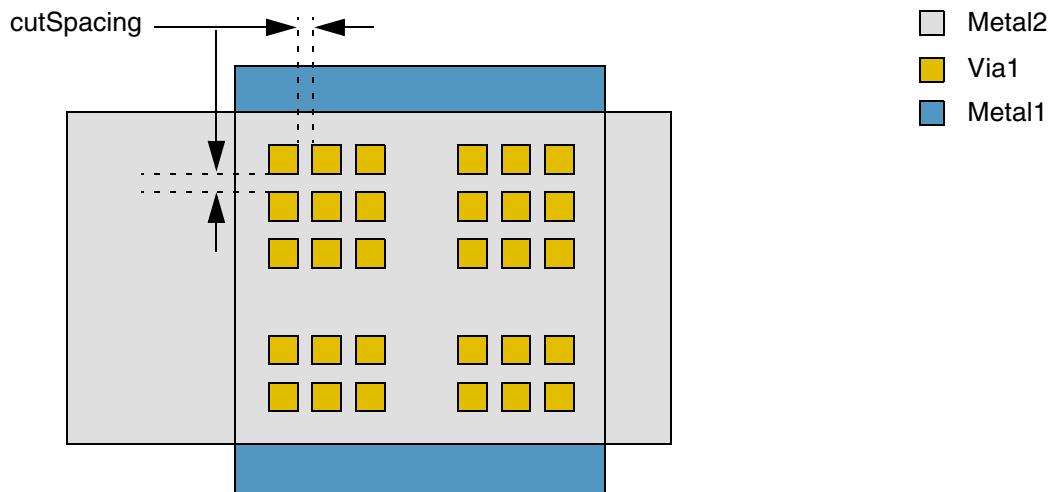
- (0.5 0.6) between two VA via cuts
- (0.2 0.2) between VA and VB via cuts
- (0.3 0.3) between VA and VC via cuts

```
orderedspacings(
    minEndOfLineCutSpacing "Via1" "Metal12"
        '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 )
)
) ;orderedspacings
```

## minLargeViaArrayCutSpacing

```
spacings(  
  ( minLargeViaArrayCutSpacing tx_cutLayer  
    ['minNumCuts x_minNumCuts] ['maxNumCuts x_maxNumCuts]  
    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 requirement 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*

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
```

(Advanced Nodes Only) The constraint applies only if the number of rows and columns in the via cut array is less than or equal to this value.

## Examples

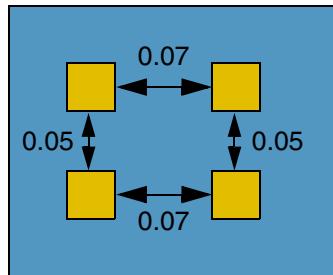
- [Example 1a: minLargeViaArrayCutSpacing with minNumCuts and maxNumCuts](#)
- [Example 1b: minLargeViaArrayCutSpacing with minNumCuts and maxNumCuts](#)

### ***Example 1a: minLargeViaArrayCutSpacing with minNumCuts and maxNumCuts***

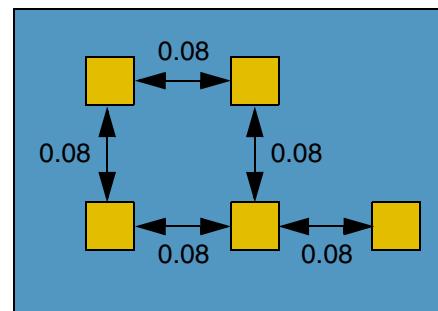
The spacing between the via cuts in a 2x2 via cut array must be greater than or equal to 0.07.

```
spacings(
  ( minLargeViaArrayCutSpacing "Vial"
    'minNumCuts 2 'maxNumCuts 2
    0.07
  )
) ;spacings
```

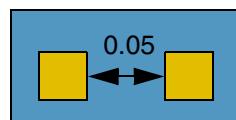
 Via1  
 Metal1



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).



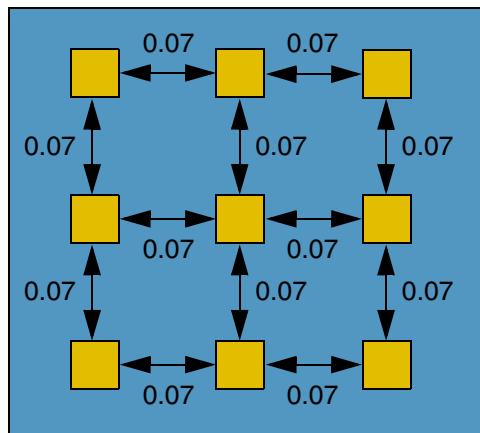
c) The constraint does not apply because this is a 1x2 via cut array (and not 2x2).

**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.

## minLargeViaArraySpacing

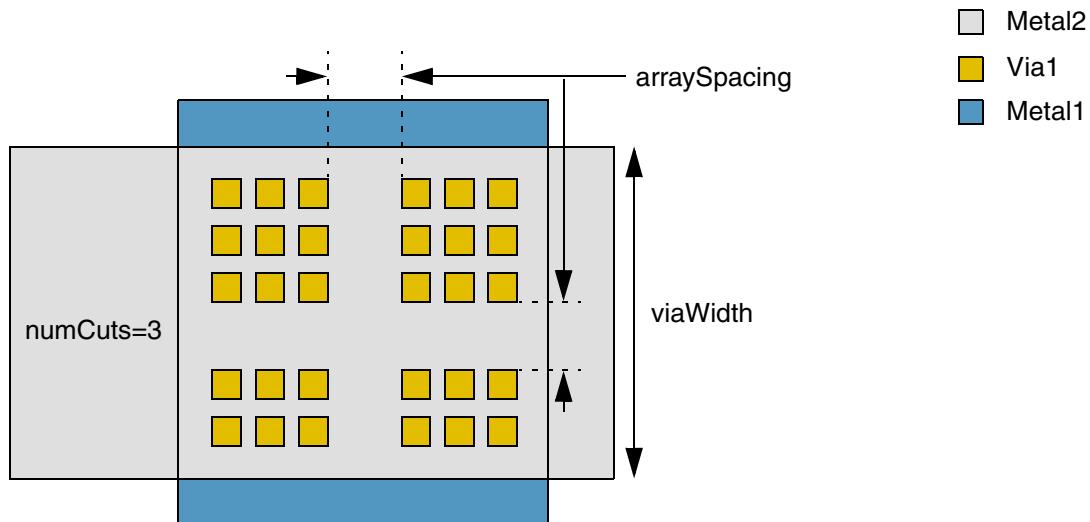
```

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*.

## Values

*tx\_cutLayer*      The cut layer on which the constraint is applied.

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

"numCuts" nil nil

This identifies the index for *table*.

*g\_table*      The format of a 1-D table is as follows:

(*x\_numCuts f\_spacing*

...

)

where, *x\_numCuts* represents the number of via cuts and *f\_spacing* 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

'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

'paraOverlap      The constraint applies only if the via cut arrays have a non-zero parallel overlap.

Type: Boolean

'intraArrayCutSpacing *f\_cutSpacing*

The via cuts that are spaced by less than this value are considered to be part of the same via cut array.

*f\_default*

The array spacing value to be used when no table entry applies.

## Examples

- [Example 1: minLargeViaArraySpacing](#)
- [Example 2: minLargeViaArraySpacing with intraArrayCutSpacing](#)

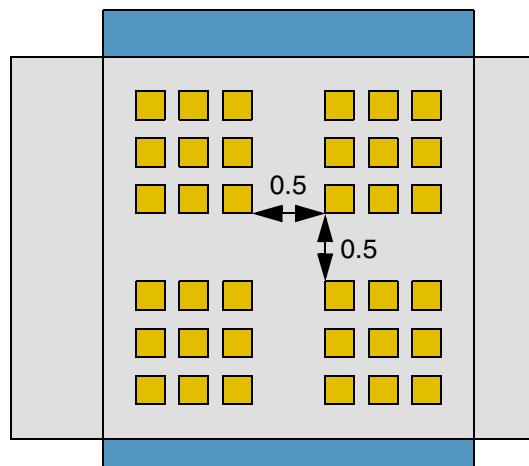
### ***Example 1: minLargeViaArraySpacing***

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 "Vial1"
        (( "numCuts" nil nil ))
        (
            2      0.4
            3      0.5
            4      0.6
            5      0.7
        )
    )
) ;spacingTables
```

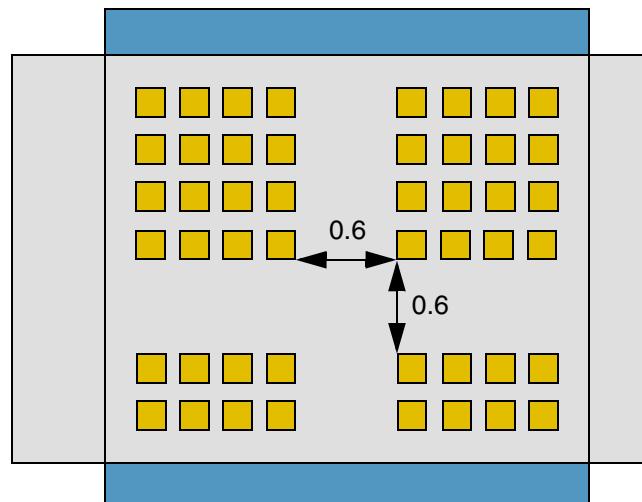
	Metal2
	Via1
	Metal1



a) PASS. For a 3x3 via array, the required array spacing is at least 0.5, a condition that is satisfied.

## Virtuoso Technology Data Constraint 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***

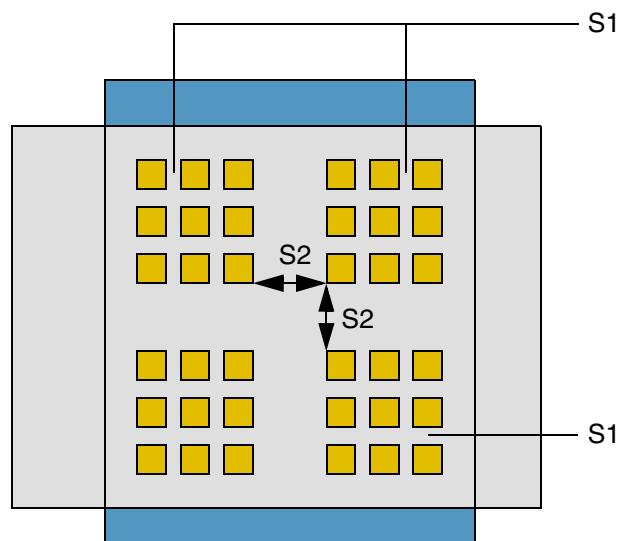
The minimum distance between via cut arrays in a large via must be as follows:

- 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  $S2 \geq v3$ .

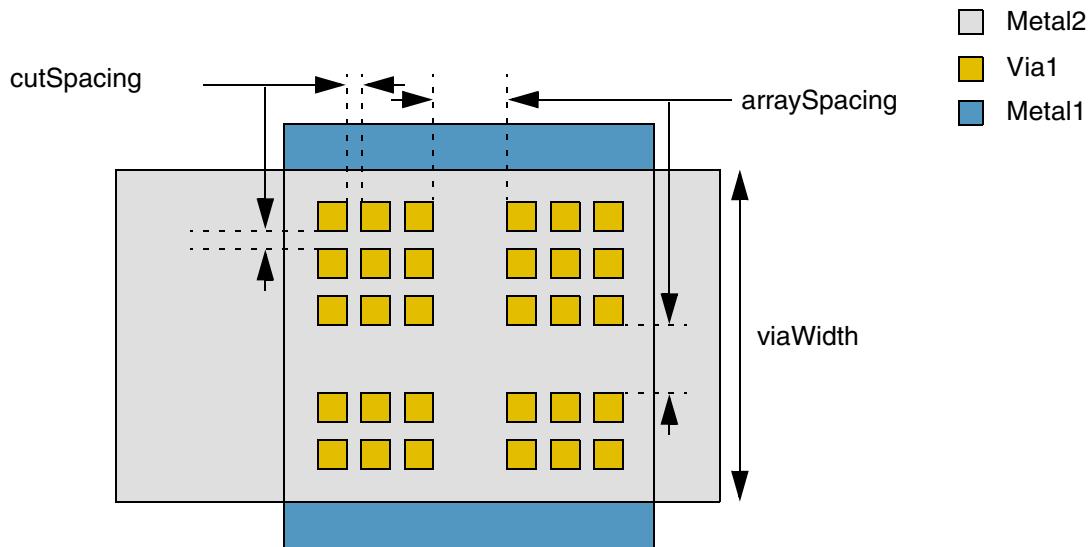
## minLargeViaArrayWidth

```
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.



## 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

'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

'paraOverlap  
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 Constraint Reference

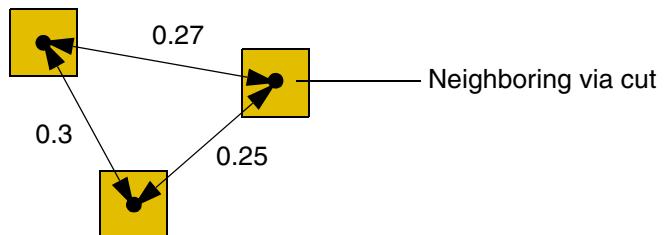
### Via Construction Constraints

#### 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



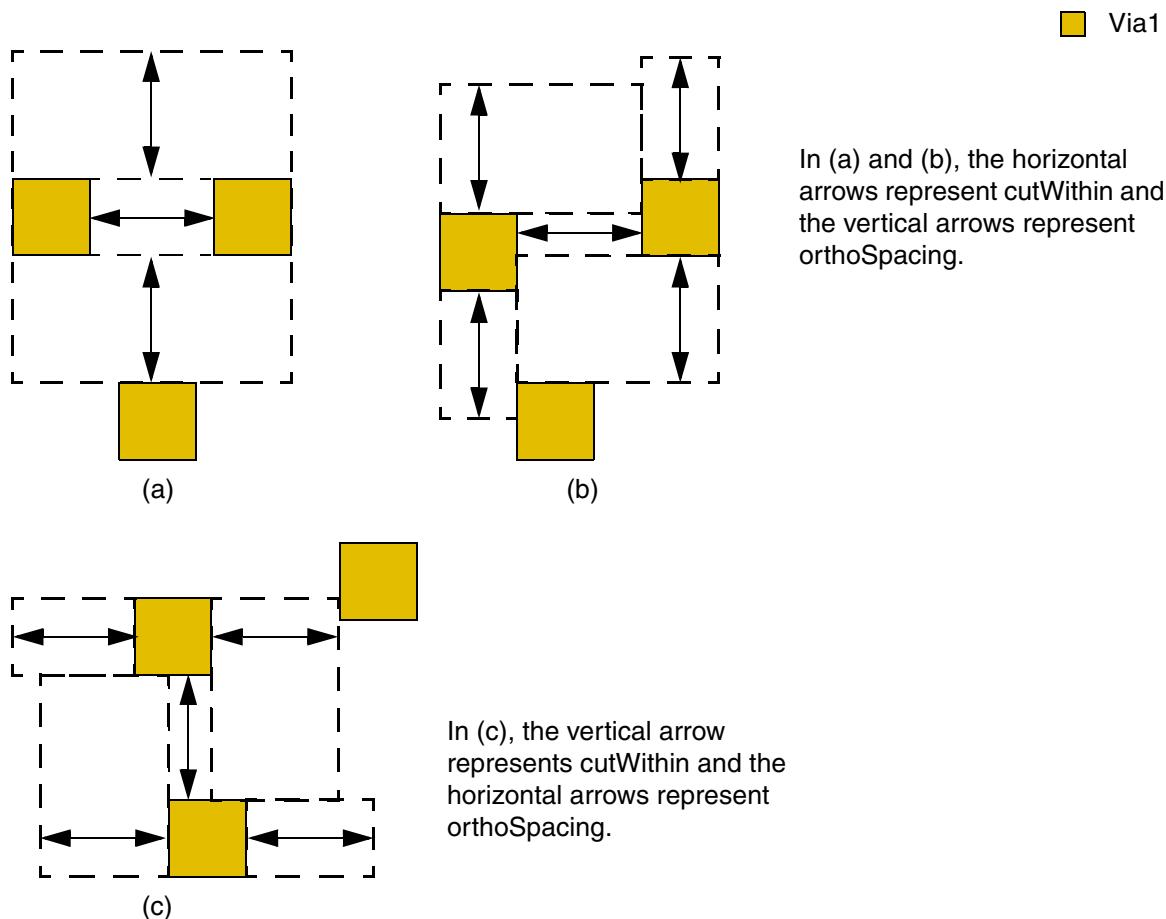
PASS. The two via cuts have a neighboring via cut at a distance less than 0.3 away and are spaced 0.3 apart (>0.2).

## 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*.



## Values

*tx\_cutLayer*      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 *table*.

*g\_table*      The format of a *table* row is as follows:

*f\_cutWithin* *f\_orthoSpacing*

where, *f\_cutWithin* is the distance between the overlapping via cuts and *f\_orthoSpacing* 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 *f\_cutWithin*.

Type: A 1-D table specifying floating-point distance and spacing values.

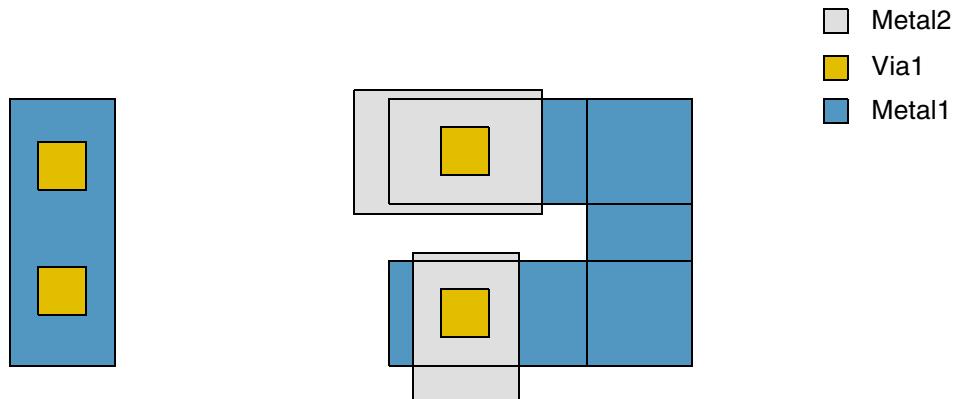
## Parameters

*f\_default*      The spacing value to be used when no table entry applies.

## minParallelViaSpacing (One layer)

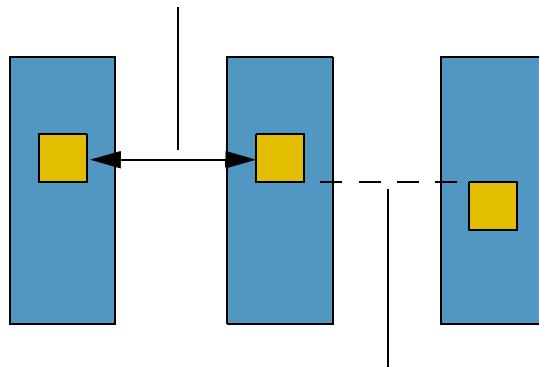
```
spacings(
  ( minParallelViaSpacing tx_layer
    ['centerToCenter']
    ['exceptSameNet | 'exceptSameMetal | 'exceptSameVia]
    f_spacing
  )
) ;spacings
```

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 Constraint 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.

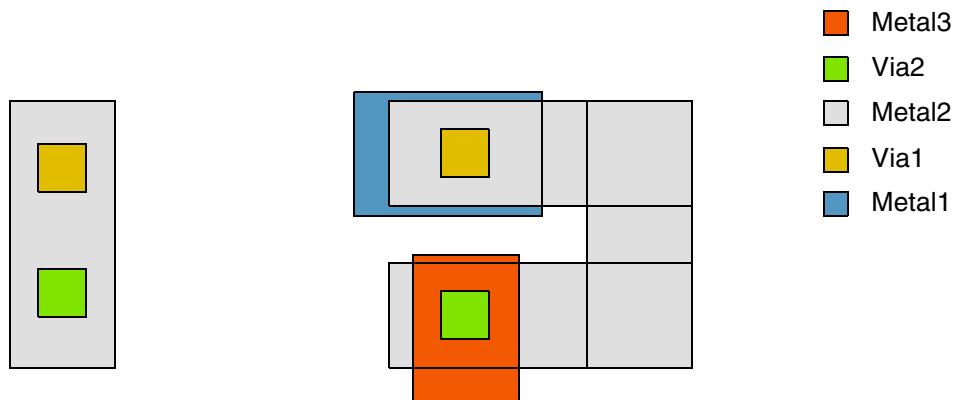
#### Parameters

'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)**

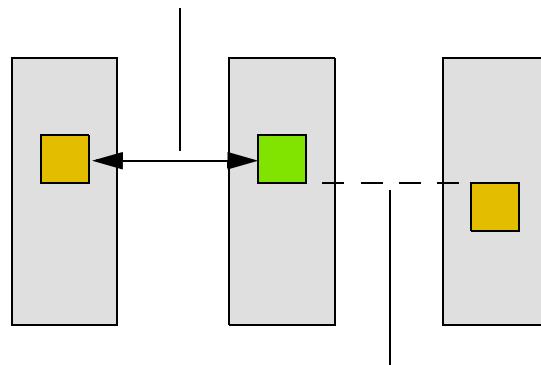
```
spacings(
  ( minParallelViaSpacing tx_cutLayer1 tx_cutLayer2 tx_metalLayer
    f_spacing
  )
) ;spacings
```

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 Constraint 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.

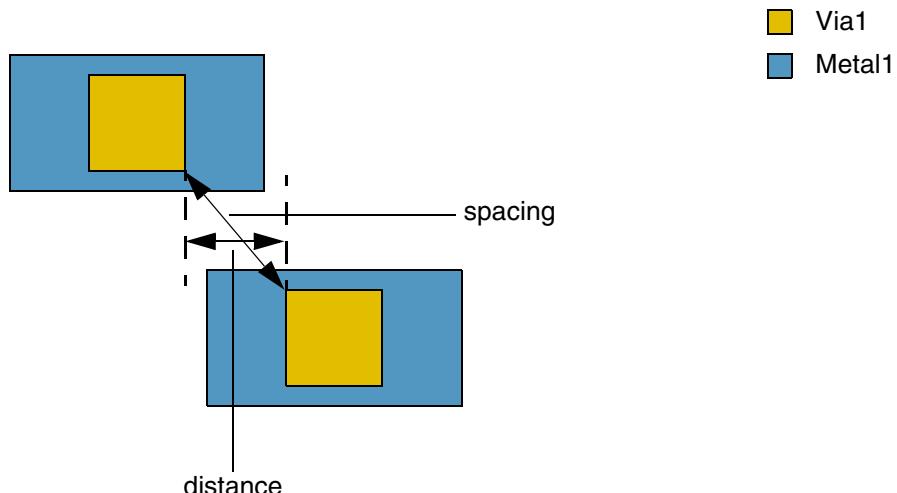
#### Parameters

None

## **minParallelWithinViaSpacing**

```
spacings(
  ( minParallelWithinViaSpacing tx_cutLayer
    ['centerToCenter']
    ['exceptSameNet']
    ['cutClass { f_width | (f_width f_length) | t_name }
      ['longEdgeOnly
        | ['otherExtension f_extension 'layer tx_metalLayer
          'paraLength f_parLength 'within f_within
        ]
      ]
    ]
    'cutDistance f_distance
    f_spacing
  )
)
; spacings
```

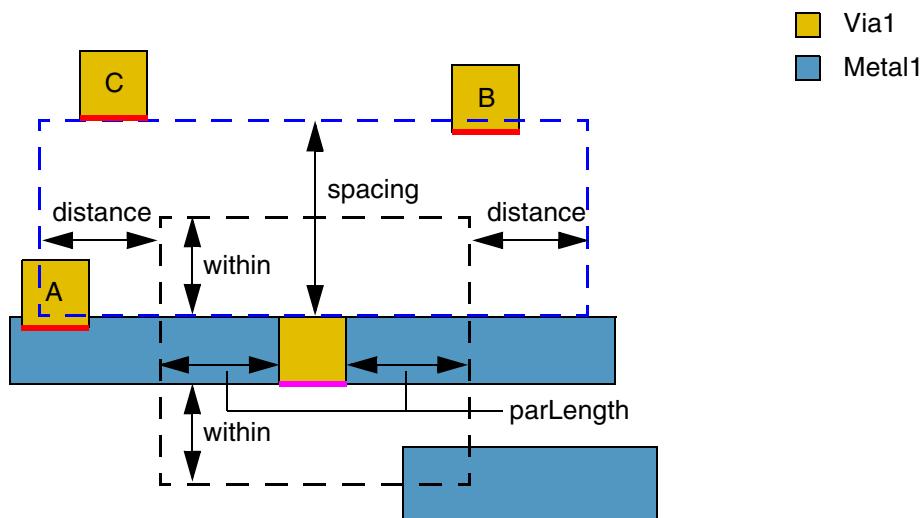
Specifies the minimum spacing required between cut shapes if the distance between the cut shapes is less than *distance*.



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.

Optionally, the constraint applies only if a cut shape has a *meta1Layer* extension less than *extension* on a side on which lies a neighboring *meta1Layer* 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.



## 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 f_length</i> )   <i>t_name</i> }	(Advanced Nodes Only) The cut class to which the constraint applies, specified by width, by width and length, or by name (as defined in a <u>cutClasses</u> constraint). <ul style="list-style-type: none"><li>■ <i>f_width</i>: Width</li><li>■ <i>f_length</i>: Length</li><li>■ <i>t_name</i>: Name of the cut class</li></ul>
'longEdgeOnly	(Advanced Nodes Only) 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

```
'otherExtension f_extension 'layer tx_metalLayer
'paraLength f_parLength '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 *parLength* 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 (*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.

```
'cutDistance f_distance
```

The constraint applies only if the distance between the cut shapes is less than this value.

## Examples

- [Example 1: minParallelWithinViaSpacing with cutClass and cutDistance](#)
- [Example 2: minParallelWithinViaSpacing with cutClass, longEdgeOnly, and cutDistance](#)

## Virtuoso Technology Data Constraint Reference

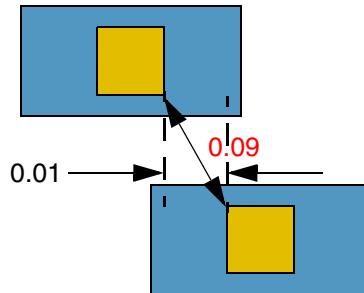
### Via Construction Constraints

#### **Example 1: minParallelWithinViaSpacing with cutClass and cutDistance**

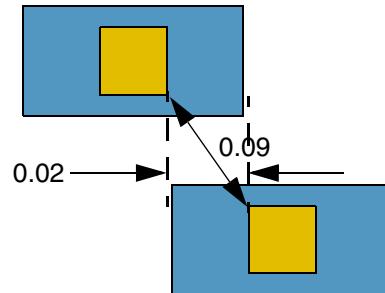
The spacing between the edges of Via1 via cuts must be at least 0.1 if the distance between two via cuts is less than 0.02.

```
spacings(
  ( minParallelWithinViaSpacing "Via1"
    'cutDistance 0.02
    0.1
  )
) ;spacings
```

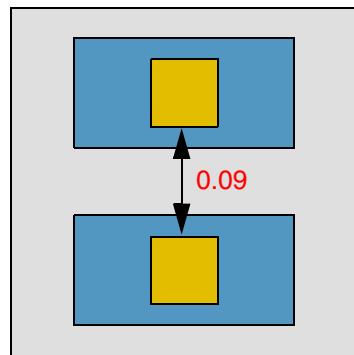
- Metal2
- Via1
- Metal1



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 Constraint Reference

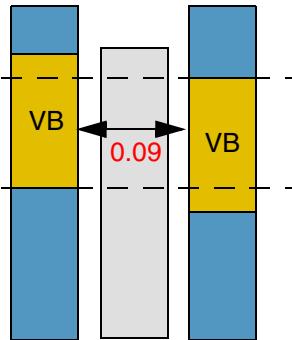
### Via Construction Constraints

#### **Example 2: minParallelWithinViaSpacing with cutClass, longEdgeOnly, and cutDistance**

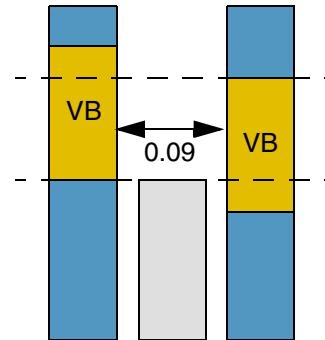
The spacing between the long edges of Via1 via cuts of type VB (0.4x0.6) must be at least 0.1 if the distance between two via cuts is less than 0.02.

```
spacings(
  ( minParallelWithinViaSpacing "Via1"
    'cutClass VB
    'longEdgeOnly
    'cutDistance 0.02
    0.1
  )
) ;spacings
```

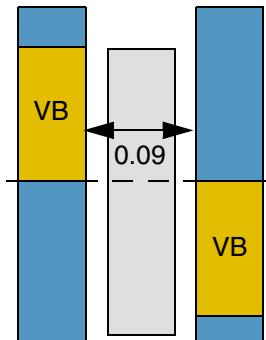
■	Metal2
■	Via1
■	Metal1



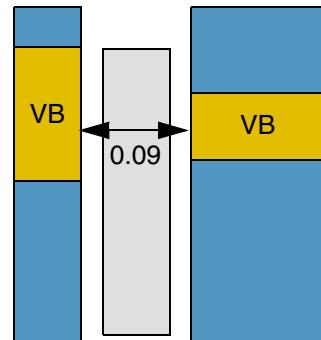
a) FAIL. The metal shape that lies between the via cuts is inside the common projected parallel run length of the via cuts, but the spacing between the via cuts is only 0.09 (<0.1).



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 is  
'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**

*tx\_cutLayer*      The cut layer on which the constraint is applied.

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

*f\_spacing*      The minimum spacing between the via cuts.

### **Parameters**

'within *f\_within*      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 Constraint 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 Constraint Reference

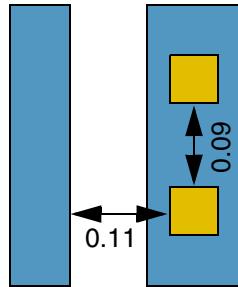
### Via Construction Constraints

#### **Example 1: minSameMetalSharedEdgeViaSpacing with within and exceptTwoEdges**

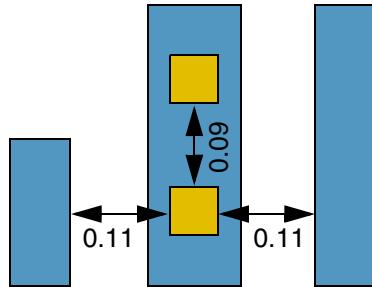
Two via cuts with a parallel overlap greater than zero and neighboring wires at a distance less than 0.12 on the same edge must have a spacing greater than or equal to 0.1.

```
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 Constraint Reference

### Via Construction Constraints

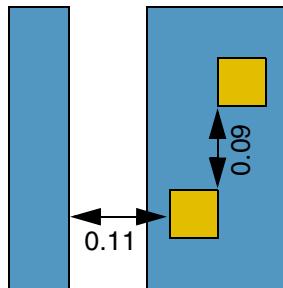
---

#### **Example 2: minSameMetalSharedEdgeViaSpacing with within**

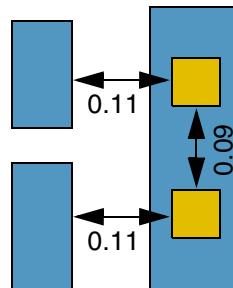
Two via cuts with a parallel overlap greater than zero and neighboring wires at a distance less than 0.12 must have a spacing greater than or equal to 0.1.

```
spacings(
  ( minSameMetalSharedEdgeViaSpacing "Via1"
    'within 0.12
    0.1
  )
) ;spacings
```

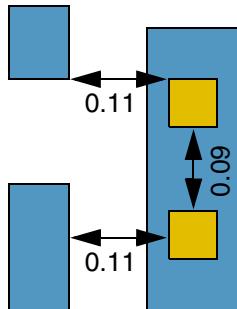
- Metal2
- Via1
- Metal1



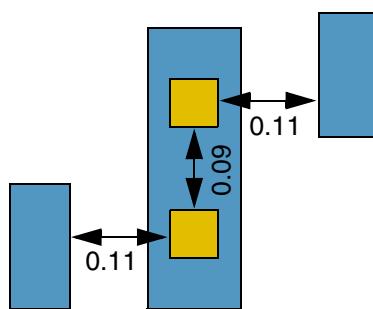
a) The constraint does not apply because the via cuts do not have a parallel overlap greater than zero.



b) FAIL. The neighbors can be different wires. However, the spacing between the via cuts is only 0.09 (<0.1).



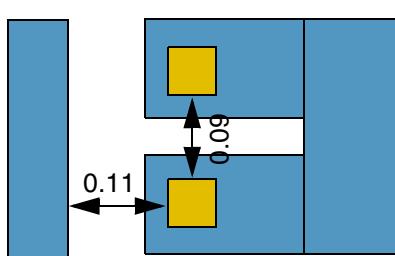
c) The constraint does not apply because the top via cut does not have a neighboring wire.



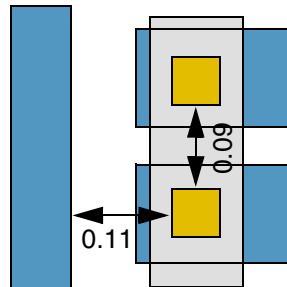
d) FAIL. Each via cut has a neighboring wire at a distance 0.11 (<0.12) from it. However, the spacing between the via cuts is only 0.09 (<0.1).

## Virtuoso Technology Data Constraint 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).

## minViaSpacing (One layer)

```

spacings(
  ( minViaSpacing tx_cutLayer
    ['centerToCenter']
    ['horizontal | 'vertical]
    ['sameNet | 'sameMetal | 'sameVia]
    ['sameMask | 'diffMask]
    ['area f_area]
    ['overLayer tx_layer ['overLayerWidth f_layerWidth]]
    ['length f_length | 'exactLength f_length ['exactSpacing]]
    ['cutClass { f_width | (f_width f_length) | t_name }]
    ['enclosingLayer tx_enclosingLayer
      ['viaEdgeType x_edgeType ['bothCuts]
       ['sizeByTouchedCorners (f_sizeBy1 f_sizeBy2)]]]
    ]
    ['sizeBy f_sizeBy]
    ['exceptExactAligned]
    f_spacing ['manhattan]
  )
)

;spacings

spacingTables(
  ( minViaSpacing tx_cutLayer
    (( "width" nil nil ["width" nil nil] )
     ['centerToCenter']
     ['horizontal | 'vertical]
     ['sameNet | 'sameMetal | 'sameVia]
     ['sameMask | 'diffMask]
     ['area f_area]
     ['overLayer tx_layer ['overLayerWidth f_layerWidth]]
     ['length f_length | 'exactLength f_exactLength ['exactSpacing]]
     ['cutClass { f_width | (f_width f_length) | t_name }]
     ['enclosingLayer tx_enclosingLayer
       ['viaEdgeType x_edgeType ['bothCuts]
        ['sizeByTouchedCorners (f_sizeBy1 f_sizeBy2)]]]
     ]
     ['sizeBy f_sizeBy]
     ['exceptExactAligned]
     ['above | 'below | 'bothAboveBelow]
     [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

*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 independent of width.

"width" nil nil

This identifies the index for *table*.

*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 Constraint Reference

## Via Construction Constraints

---

### Parameters

'centerToCenter      The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge.

Type: Boolean

'horizontal | 'vertical

(Advanced Nodes Only) The direction in which the constraint applies. If direction is not specified, the constraint applies in any direction.

Type: Boolean

'sameNet | 'sameMetal | 'sameVia

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.
- '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

'sameMask | 'diffmask

The mask type. If mask type is not specified, the constraint applies to all cuts on the layer.

- 'sameMask: (Advanced Nodes Only) The constraint applies only to via cuts on the same mask.
- 'diffMask: (ICADV12.3 Only) 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 Constraint Reference

### Via Construction Constraints

---

'overLayer *tx\_layer*

(Advanced Nodes Only) The constraint applies only if both via cuts overlap a single shape on this layer.

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

'overLayerWidth *f\_layerWidth*

(Advanced Nodes Only) The constraint applies only if the width of the overlapping shape is less than or equal to this value.

'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.
$\geq -1$ and $<-0.05$	-1.0
$\geq -0.05$ and $<0$	-0.05
$\geq 0$ and $<0.05$	0
$\geq 0.05$ and $<1.0$	0.05
$\geq 1.0$	1.0

'exactLength *f\_exactLength*

The constraint applies only if the parallel run length between the two via cuts is exactly equal to this value.

'exactSpacing

The spacing between the via cuts must be equal to the constraint value.

Type: Boolean

## Virtuoso Technology Data Constraint 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

'enclosingLayer *tx\_enclosingLayer*

(Advanced Nodes Only) The enclosing layer for  
'viaEdgeType extension checks.

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

'viaEdgeType *x\_edgeType*

(Advanced Nodes Only) The constraint applies only to the via cut edges of this type (the type is defined in the viaEdgeType constraint).

'bothCuts

(Advanced Nodes Only) The constraint applies only if both via cuts satisfy 'viaEdgeType.

Type: Boolean

'sizeByTouchedCorners (*f\_sizeBy1 f\_sizeBy2*)

(ICADV12.3 Only) 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.

'sizeBy *f\_sizeBy*

(Advanced Nodes Only) 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.

'exceptExactAligned

(Advanced Nodes Only) The constraint applies only to those via cuts that are not exactly aligned.

Type: Boolean

'above | 'below | 'bothAboveBelow

(ICADV12.3 Only) 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.

*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.

Type: Boolean

## Examples

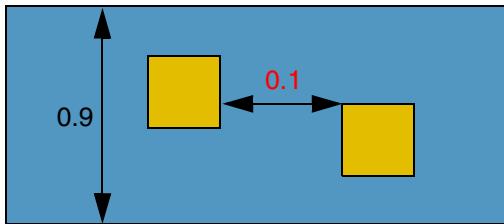
- [Example 1: minViaSpacing with overLayer and overLayerWidth](#)
- [Example 2: minViaSpacing with centerToCenter, enclosingLayer, and viaEdgeType](#)
- [Example 3: minViaSpacing with exactLength, enclosingLayer, viaEdgeType, bothCuts, and sizeBy](#)
- [Example 4: minViaSpacing with exceptExactAligned](#)
- [Example 5: minViaSpacing with centerToCenter, cutClass, enclosingLayer, and sizeByTouchedCorners](#)

**Example 1: minViaSpacing with overLayer and overLayerWidth**

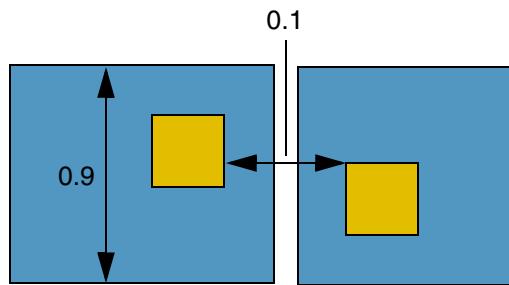
The spacing between two via cuts must be at least 0.2 if they overlap a Metal1 shape with width less than or equal to 0.09.

```
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).



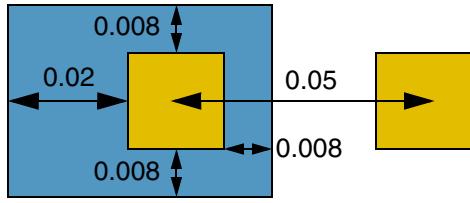
b) The constraint does not apply because the two via cuts do not overlap the same Metal1 shape.

**Example 2: *minViaSpacing* with *centerToCenter*, *enclosingLayer*, and *viaEdgeType***

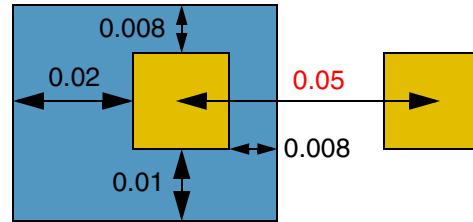
The center-to-center spacing between two via cuts must be at least 0.06. The constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.009.

```
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.



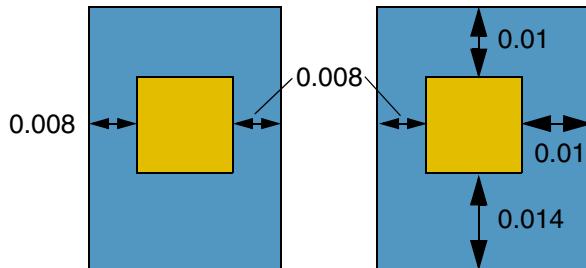
- b) FAIL. There are no opposite edges with Metal1 extension less than or equal to 0.009, but the spacing between the two cut vias is 0.05 (<0.06).

**Example 3: *minViaSpacing* with *exactLength*, *enclosingLayer*, *viaEdgeType*, *bothCuts*, and *sizeBy***

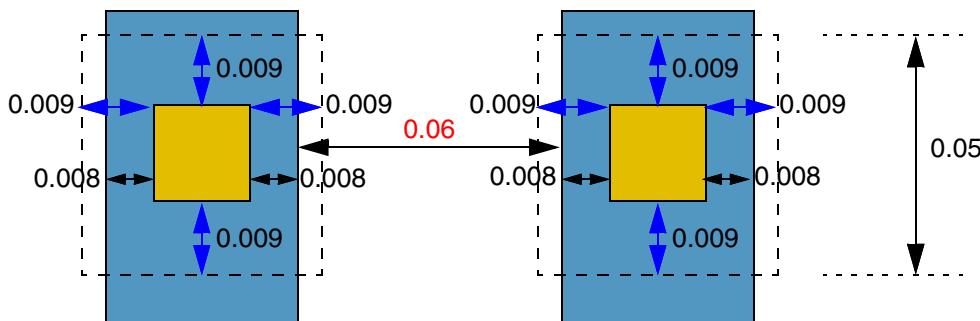
The spacing between two via cuts expanded by 0.009 must be at least 0.07 if both via cuts have Metal1 extensions less than or equal to 0.009 on a pair of opposite edges and the parallel run length between them is 0.05.

```
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.



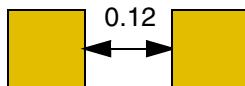
- b) FAIL. Both via cuts have a pair of opposite edges with 0.008 Metal1 extension (<0.009), and the parallel run length between the via cuts expanded by 0.009 is 0.05. However, the spacing between the enclosing metal edges (the AND of the Metal1 enclosing layer extension and the via cut expanded by 0.009) is 0.06 (<0.07).

**Example 4: minViaSpacing with exceptExactAligned**

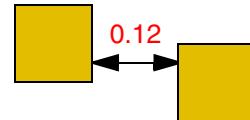
The spacing must be greater than or equal to 0.1 between exactly aligned via cuts and greater than or equal to 0.15 between via cuts that are not exactly aligned.

```
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).



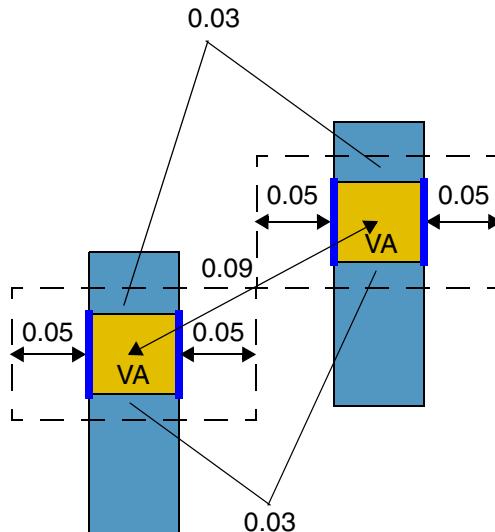
b) FAIL. The cuts are not exactly aligned, but the spacing between them is 0.12 (<0.15).

**Example 5: 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)
    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).

## 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 }']
    ['enclosingLayer tx_enclosingLayer
      ['viaEdgeType x_edgeType ['extendBy f_extendBy]
    ]
    ['sizeBy f_sizeBy]
    ['exceptSameNet | 'exceptSameMetal]
    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]
    [f_default]
  )
  (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.

## 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:  $(f\_widthCut1\ f\_spaceValue) \text{ or } (f\_widthCut1\ f\_widthCut2\ f\_spaceValue)$ where, <ul style="list-style-type: none"><li>■ <i>f_widthCut1</i>: The width of the via cut on <i>cutlayer1</i> must be greater than or equal to this value.</li><li>■ <i>f_widthCut2</i>: The width of the via cut on <i>cutlayer2</i> must be greater than or equal to this value.</li><li>■ <i>f_spaceValue</i>: The spacing between the two cuts on different layers must be greater than or equal to this value.</li></ul> 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 Constraint Reference

### Via Construction Constraints

---

#### Parameters

'centerToCenter      The spacing is measured center-to-center. Otherwise, the spacing is measured edge-to-edge.

Type: Boolean

'sameNet | 'sameMetal

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.

Type: Boolean

'stack

The via cuts on two different layers can be stacked if their centers are exactly aligned. Otherwise, the via cuts cannot be stacked.

'area *f\_area*

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.

'cutClass { *f\_width* | (*f\_width f\_length*) | *t\_name* }

(Advanced Nodes Only) The cut class to which the constraint applies on *cutLayer1*, specified by width, by width and length, or by name (as defined in a cutClass constraint).

- *f\_width*: Width
- *f\_length*: Length
- *t\_name*: Name of the cut class

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints

---

'otherCutClass { *f\_width* | (*f\_width f\_length*) | *t\_name* }

(Advanced Nodes Only) 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

'enclosingLayer *tx\_enclosingLayer*

(Advanced Nodes Only) The enclosing layer for 'viaEdgeType extension checks.

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

'viaEdgeType *x\_edgeType*

(Advanced Nodes Only) 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*

(Advanced Nodes Only) 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*

(Advanced Nodes Only) 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 Constraint Reference

### Via Construction Constraints

---

'exceptSameNet | 'exceptSameMetal

(ICADV12.3 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

*f\_default*

The spacing value to be used when no table entry applies.

### Examples

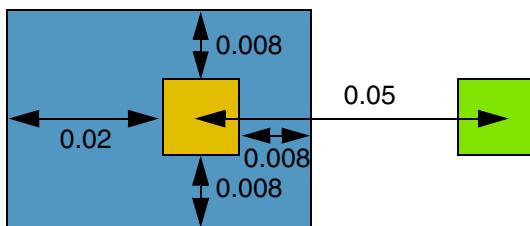
- [Example 1: minViaSpacing with centerToCenter, enclosingLayer, and viaEdgeType](#)
- [Example 2: minViaSpacing with edgeType, enclosingLayer, sizeBy, and extendBy](#)

**Example 1: minViaSpacing with centerToCenter, enclosingLayer, and viaEdgeType**

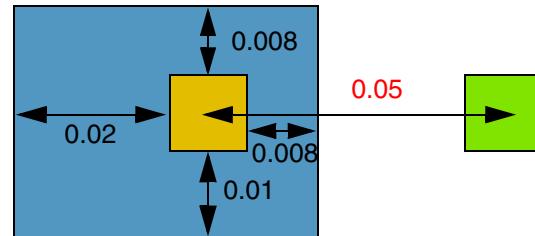
The center-to-center spacing between a Via1 cut that satisfies viaEdgeType 1 (the constraint applies only to via cuts that do not have two opposite edges with extensions less than or equal to 0.009) and a Via2 cut must be at least 0.06.

```
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).



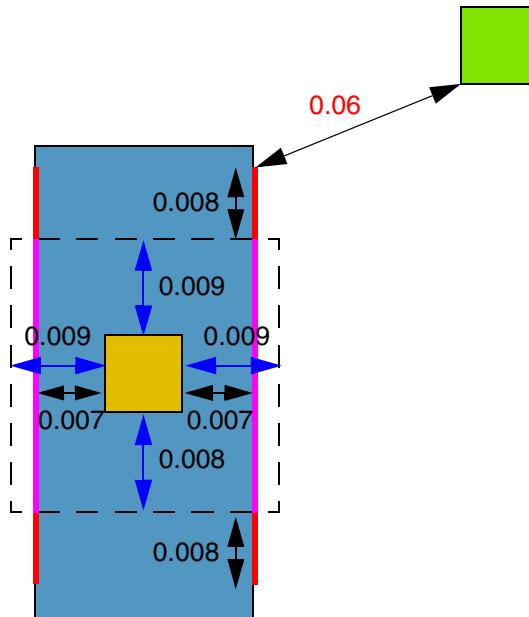
- b) FAIL. The constraint applies because the edges with extension 0.008 (that is, with extension less than or equal to 0.009) are not opposite edges. However, the center-to-center spacing between the two via cuts is 0.05 (<0.06).

**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

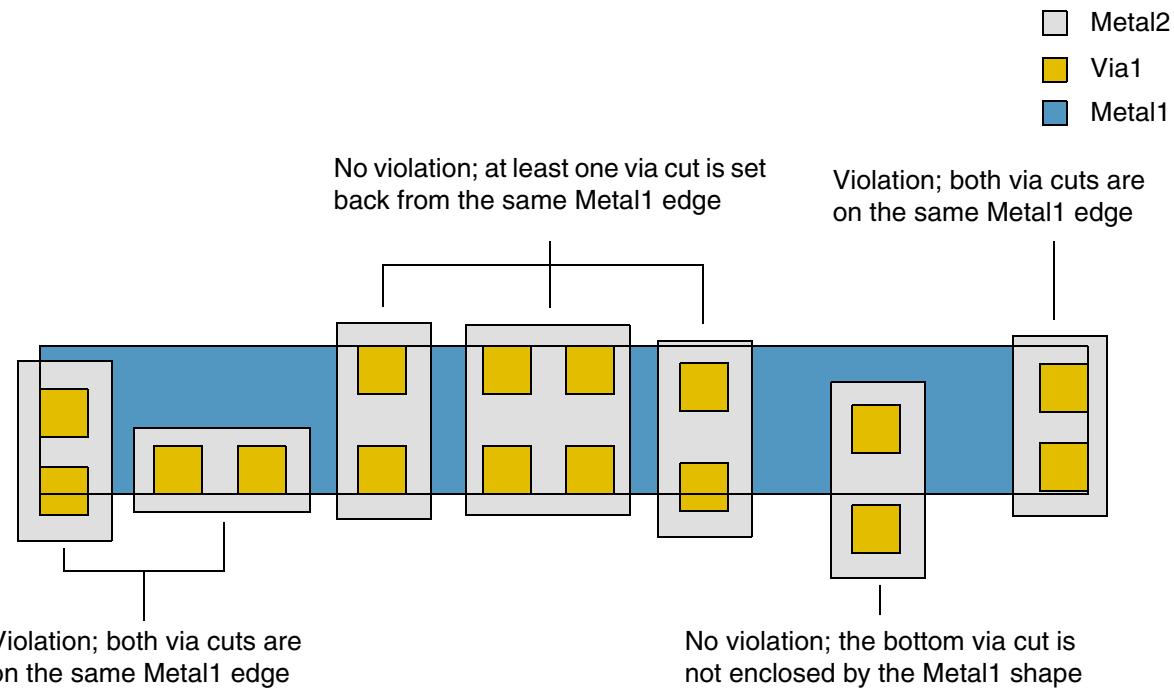


FAIL. The constraint applies because the Via1 cut has a pair of opposite edges (left and right) with extension 0.007 (that is, with extension less than or equal to 0.009). The pink edges indicate the intersection of Via1 cut and Metal1 shape after the via cut is sized by 0.009. The pink edges are extended 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.

## 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 Constraint 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> . 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</i>   <i>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]
    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.

### **Values**

*tx\_layer*                   The cut layer on which the constraint is applied.

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

*x\_viaEdgeType*           The type assigned to the cut edges.

### **Parameters**

'edgeExtension *f\_edgeExt*

The constraint applies to cut edges with extension less than or equal to this value.

'anyOppositeExtension *f\_oppEdgeExt*

The constraint applies only if a pair of opposite edges have extensions less than or equal to this value.

'negateEdgeExtension

The constraint applies to cut edges that do not satisfy 'edgeExtension.

'negateAnyOppositeExtension

The constraint applies only if 'anyOppositeExtension is not satisfied.

## Examples

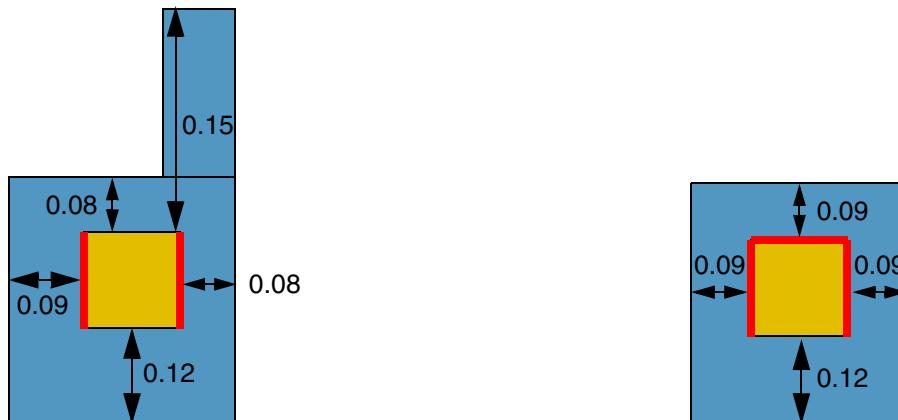
- [Example 1: viaEdgeType with edgeExtension](#)
- [Example 2: viaEdgeType with edgeExtension and anyOppositeExtension](#)
- [Example 3: viaEdgeType with edgeExtension and negateEdgeExtension](#)
- [Example 4: viaEdgeType with edgeExtension, anyOppositeExtension, and negateAnyOppositeExtension](#)

### ***Example 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 "Vial"
    '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.

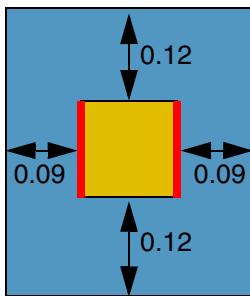
b) The three edges shown in red are of type 1 because these edges have extensions (less than or) equal to 0.09.

**Example 2: viaEdgeType with edgeExtension and anyOppositeExtension**

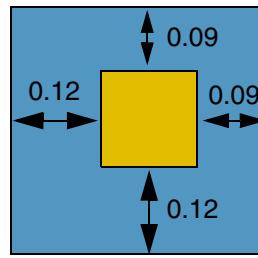
A via cut edge or a pair of opposite via cut edges are of type 2 if they have extensions of less than or equal to 0.09 along the entire length of an edge.

```
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.



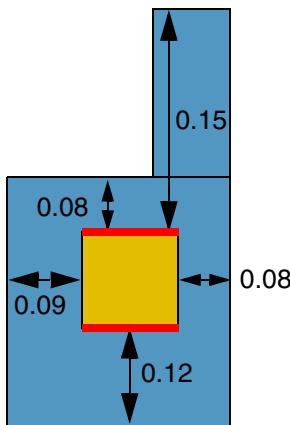
b) The constraint does not apply because the via cut does not have two opposite edges with extension less than or equal to 0.09.

**Example 3: viaEdgeType with edgeExtension and negateEdgeExtension**

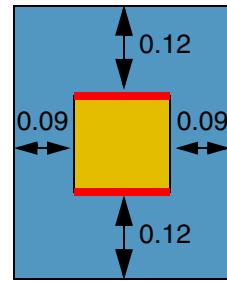
A via cut edge is of type 3 if it has an extension of greater than 0.09 along the entire length of an edge.

```
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.



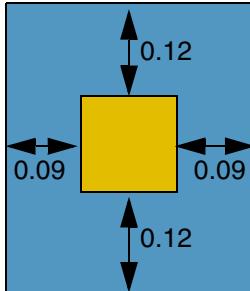
b) The top and bottom edges have extension 0.12 ( $>0.09$ ), and are, therefore, of type 3.

**Example 4: viaEdgeType with edgeExtension, anyOppositeExtension, and negateAnyOppositeExtension**

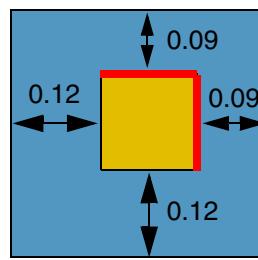
A via cut edge is of type 4 if it has an extension of less than or equal to 0.09 along the entire length of the edge. A pair of opposite via cut edges are of type 4, if they have extensions greater than 0.09 along the entire length of the edge.

```
spacings(
  ( viaEdgeType "Via1"
    'edgeExtension 0.09
    'anyOppositeExtension 0.09
    'negateAnyOppositeExtension
    2
  )
) ;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.

## **viaRequiredWithin (Advanced Nodes Only)**

```
spacings(
  ( viaRequiredWithin tx_layer
    ['cutClass { f_width | (f_width f_length) | t_name }]
    f_spacing
  )
) ;spacings
```

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**

'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 Constraint Reference

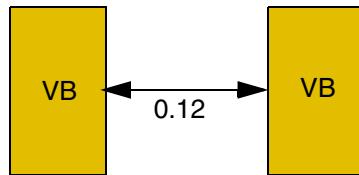
## Via Construction Constraints

### Example

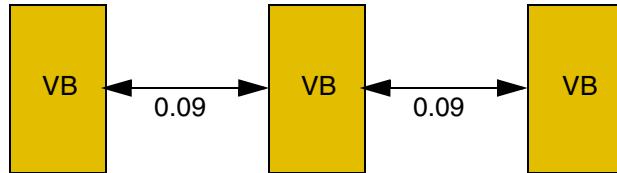
The distance between two neighboring VB via cuts on Via1 must be less than 0.1.

```
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 at spaced at 0.09 (<0.1).

## viaSpacing

```
spacings(
    ( viaSpacing tx_cutLayer
        ['centerToCenter']
        ['sameNet | 'sameMetal | 'sameMetalOverlap | 'sameVia]
        ['exceptSamePGNet]
        ['cutClass { f_width | (f_width f_length) | t_name }
            ['sideParaOverlap] ['allCuts]
        ]
        ['exactAligned x_numAlignedCuts ['exactSpacing] | 'noPrl]
        ['cutSizeRanges (f_xRange f_yRange)
            | 'cutSizeRanges (f_xRange1 f_yRange1 f_xRange2 f_yRange2)
        ]
        ['exceptOppositeCornerNeighbors]
        ['twoCuts x_twoCuts ['sameNeighborCuts]]
        ['sameMask]
        ['minDistance f_lowDistance]
        ['exceptWithin f_exceptWithin]
        'numCuts x_numCuts
        'distance f_distance
        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.

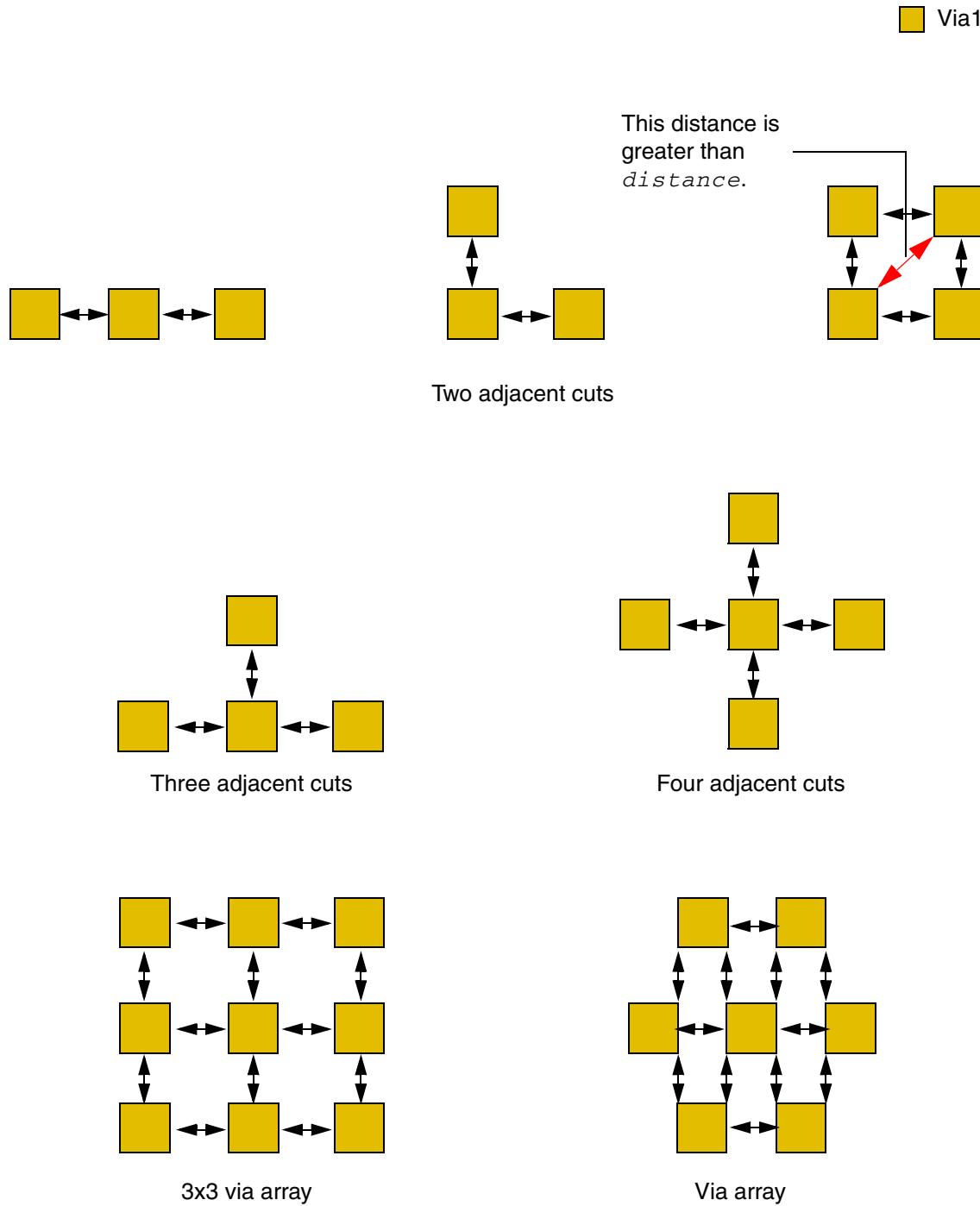
A pair of via cuts is considered adjacent if the distance between the via cuts in any direction is less than the specified *distance* value. This includes the distance measured along a 45-degree angle. Additionally, a via cut must have a certain minimum number of adjacent via cuts for the constraint 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 Constraint Reference

### Via Construction Constraints

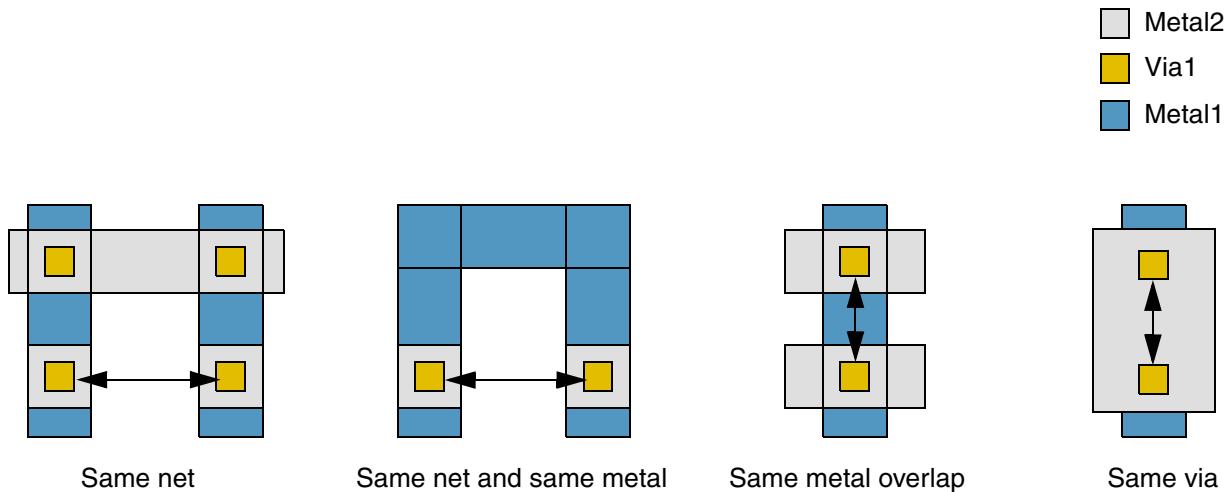
The following figures illustrate the concept of adjacency. The distance between two adjacent via cuts must be less than or equal to *distance*, denoted by the black arrows in the figures below:



# Virtuoso Technology Data Constraint Reference

## Via Construction Constraints

The following figures illustrate the different connectivity types:



## 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 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 Constraint Reference

### Via Construction Constraints

---

' sameNet | ' sameMetal | ' sameMetalOverlap | ' sameVia

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.

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

' allCuts

(Advanced Nodes Only) The neighboring via cuts can belong to any cut class.

Type: Boolean

## Virtuoso Technology Data Constraint 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

'noPrl

The constraint applies only between via cuts with parallel run length less than zero.

Type: Boolean

'cutSizeRanges (*f\_xRange f\_yRange*)

| 'cutSizeRanges (*f\_xRange1 f\_yRange1 f\_xRange2 f\_yRange2*)

(Advanced Nodes Only) 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

(Advanced Nodes Only) The constraint does not apply if the adjacent via cuts are on opposite corners.

Type: Boolean

## Virtuoso Technology Data Constraint Reference

### Via Construction Constraints

---

'twoCuts *x\_twoCuts*

(Advanced Nodes Only) 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.

'sameNeighborCuts

(ICADV12.3 Only) The via cuts that meet the 'twoCuts requirement must have the same adjacent via cuts as neighbors.

Type: Boolean

'sameMask

(ICADV12.3 Only) The constraint applies only to via cuts on the same mask.

Type: Boolean

'minDistance *f\_lowDistance*

(ICADV12.3 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*

(ICADV12.3 Only) The constraint does not apply if all adjacent via cuts are at a distance less than this value.

'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*

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.

## Old Syntax

```
spacings( ( viaSpacing tx_cutLayer g_spacingDefinition ) ... )
```

where,

*g\_spacingDefinition* is specified as *f\_spacing* or (*x\_numCuts f\_distance f\_spacing*)

## Example

```
spacings(  
    ( viaSpacing "Vial" 0.5 )  
    ( viaSpacing "Cont1" (3 1.5 1.0) )  
) ;spacings
```

## Examples

- [Example 1: viaSpacing with centerToCenter and noPrl](#)
- [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 noPrl](#)
- [Example 6: viaSpacing with cutClass, allCuts, and noPrl](#)
- [Example 7: viaSpacing with cutClass and sideParaOverlap](#)
- [Example 8: viaSpacing with cutSizeRanges and noPrl](#)
- [Example 9: viaSpacing with cutSizeRanges and exceptOppositeCornerNeighbors](#)
- [Example 10: viaSpacing with exceptWithin, numCuts, and distance](#)

# Virtuoso Technology Data Constraint Reference

## Via Construction Constraints

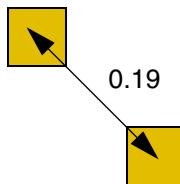
### Example 1: viaSpacing with centerToCenter and noPrl

The center-to-center spacing between neighboring via cuts must be at least 0.21 if:

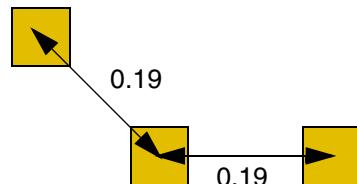
- A via cut has at least two neighboring via cuts less than 0.2 away.
- The via cuts do not have a parallel overlap.

```
spacings(
  ( viaSpacing "Via1"
    'centerToCenter
    'noPrl
    'numCuts 2
    'distance 0.2
    0.21
  )
) ;spacings
```

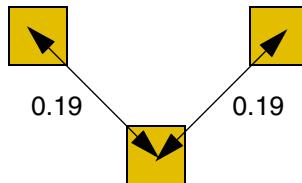
 Via1



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.

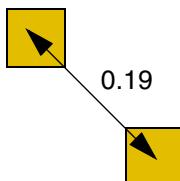
**Example 2: viaSpacing with centerToCenter and exactAligned**

The center-to-center spacing between neighboring via cuts must be at least 0.21 if:

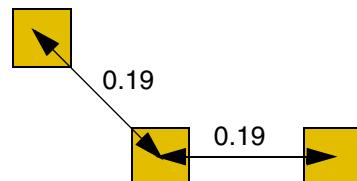
- A via cut has at least two neighboring via cuts less than 0.2 away, and at least one of these neighboring via cuts is not exactly aligned.
- There exist three exactly aligned via cuts, horizontally and/or vertically.

```
spacings(
    ( viaSpacing "Via1"
        'centerToCenter
        'exactAligned 3
        'numCuts 2
        'distance 0.2
        0.21
    )
) ;spacings
```

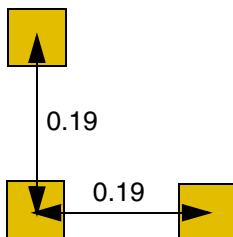
■ Via1



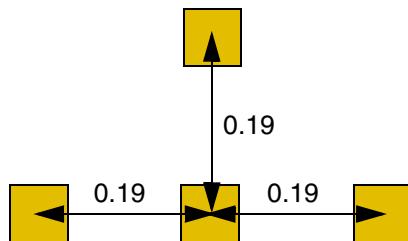
a) The constraint does not apply because there exists only one adjacent via cut.



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.



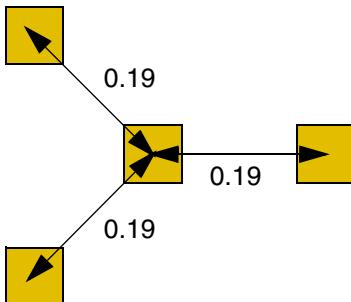
d) FAIL. There are three exactly aligned adjacent via cuts, two horizontally and one vertically. Therefore, the spacing between adjacent via cuts must be at least 0.21.

**Example 3: viaSpacing with centerToCenter and twoCuts**

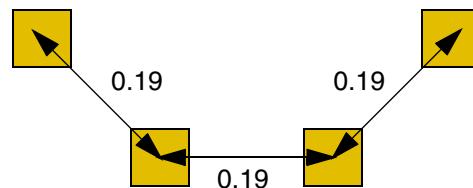
The center-to-center spacing between two neighboring via cuts must be at least 0.21 if each via cut has exactly two neighboring via cuts less than 0.2 away. Otherwise, there must be at least three adjacent via cuts.

```
spacings(
  ( viaSpacing "Via1"
    'centerToCenter
    'twoCuts 2
    'numCuts 3
    'distance 0.2
    0.21
  )
) ;spacings
```

■ Via1



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 Constraint Reference

### Via Construction Constraints

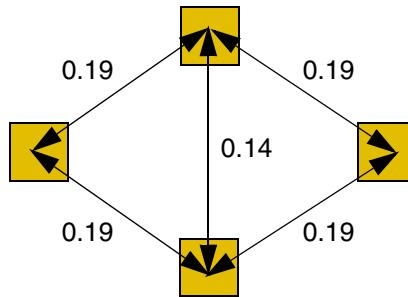
---

#### **Example 4: viaSpacing with centerToCenter, twoCuts, and sameNeighborCuts**

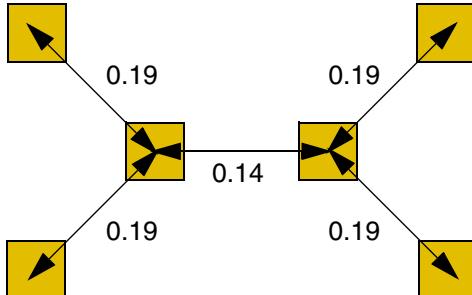
The center-to-center spacing between two neighboring via cuts must be at least 0.21 if each via cut has at least three common neighboring cuts less than 0.2 away. Otherwise, there must be at least four adjacent via cuts.

```
spacings(
  ( viaSpacing "Vial1"
    'centerToCenter
    'twoCuts 3 'sameNeighborCuts
    'numCuts 4
    'distance 0.2
    0.21
  )
) ;spacings
```

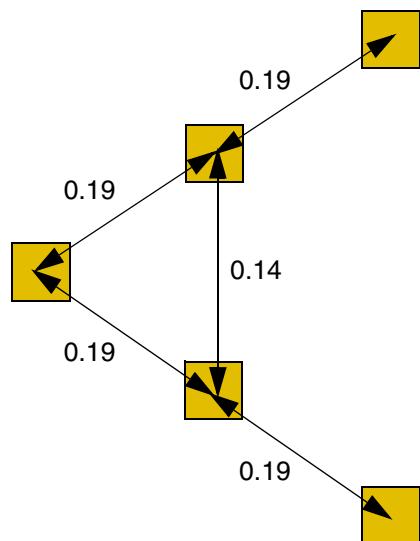
■ Via1



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.

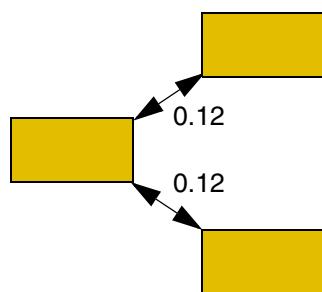
**Example 5: viaSpacing with cutClass and noPrl**

The edge-to-edge spacing between 0.1x0.2 via cuts must be at least 0.14 if:

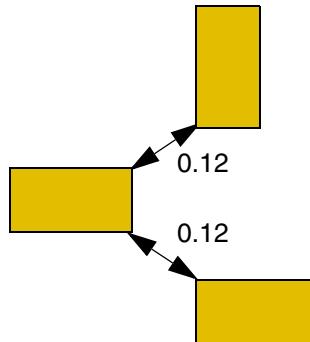
- A via cut has at least two neighboring cuts at a distance less than 0.13 away.
- The via cuts do not have a parallel overlap.

```
spacings(
  viaSpacing "Via1"
    'noPrl
    'cutClass (0.1 0.2)
    'numCuts 2
    'distance 0.13
    0.14
  )
) ;spacings
```

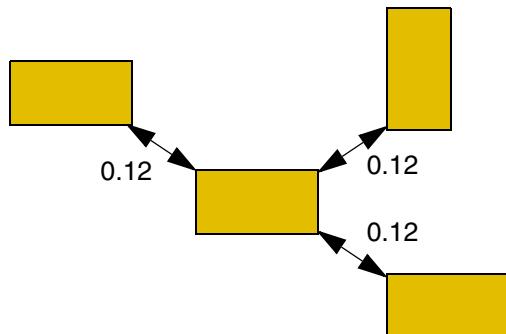
■ Via1



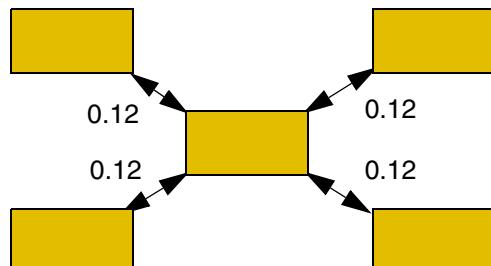
a) FAIL. The middle via cut has two adjacent via cuts that have parallel run length less than zero with it. Therefore, the spacing between them must be at least 0.14.



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.

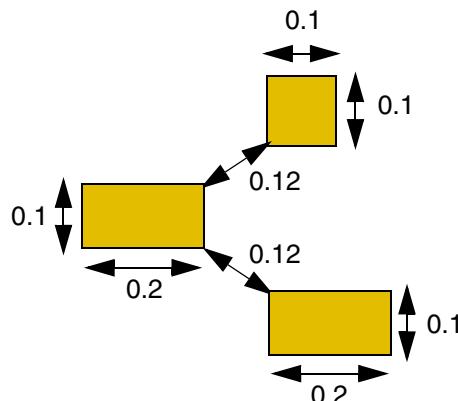
**Example 6: viaSpacing with cutClass, allCuts, and noPrl**

The edge-to-edge spacing between a 0.1x0.2 via 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.

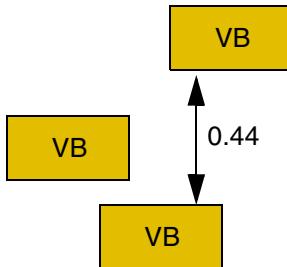
**Example 7: viaSpacing with cutClass and sideParaOverlap**

The edge-to-edge spacing between neighboring via cuts must be at least 0.45 if:

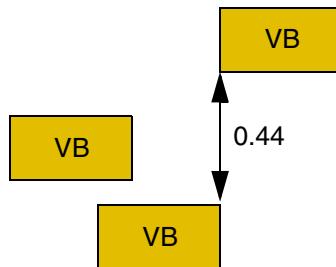
- A VB via cut has at least two neighboring VB via cuts less than 0.45 away.
- The long edges of the via cuts do not have a parallel overlap.

```
spacings(
  viaSpacing "Via1"
    'cutClass ("VB")
    'sideParaOverlap
    'numCuts 2
    'distance 0.45
    0.45
)
) ;spacings
```

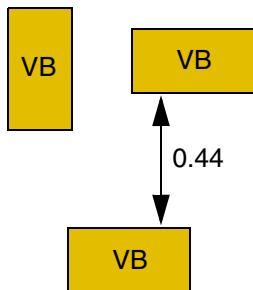
■ Via1



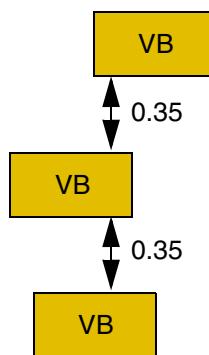
a) FAIL. The long edges of the via cuts have parallel run length greater than zero. Therefore, the spacing between the via cuts must be at least 0.45.



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.

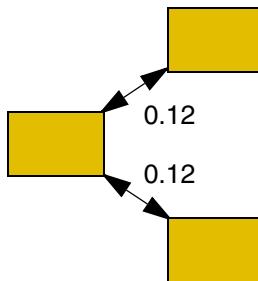
**Example 8: viaSpacing with cutSizeRanges and noPrl**

The edge-to-edge spacing between via cuts must be at least 0.14 if:

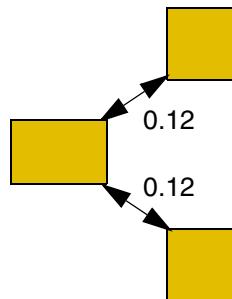
- The first via cut is 0.15 xSpan x 0.1 ySpan and the neighboring via cuts are 0.1 xSpan x 0.1 ySpan.
- The first via cut has at least two neighboring via cuts less than 0.13 away
- The via cuts do not have a parallel overlap.

```
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.

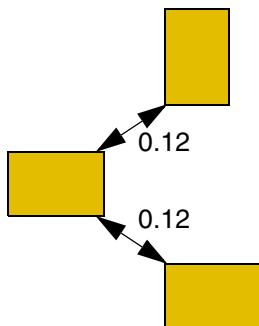
**Example 9: viaSpacing with cutSizeRanges and exceptOppositeCornerNeighbors**

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:

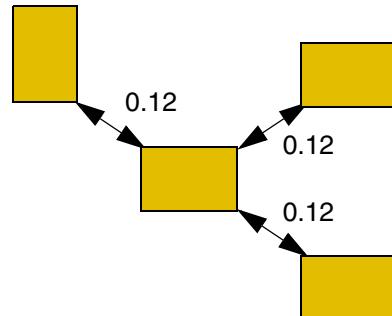
- A via cut has at least two neighboring via cuts less than 0.13 away.
- The adjacent via cuts are not on opposite corners.

```
spacings(
    ( viaSpacing "Vial"
        'cutSizeRanges (0.15 0.1)
        'exceptOppositeCornerNeighbors
        'numCuts 2
        'distance 0.13
        0.14
    )
) ;spacings
```

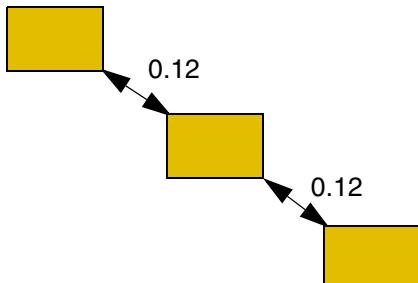
■ Via1



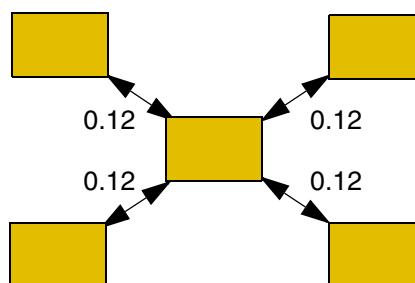
a) The constraint does not apply because only one adjacent via cut exists. The top via cut is not counted as adjacent because its dimensions are 0.1 xSpan x 0.15 ySpan.



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.



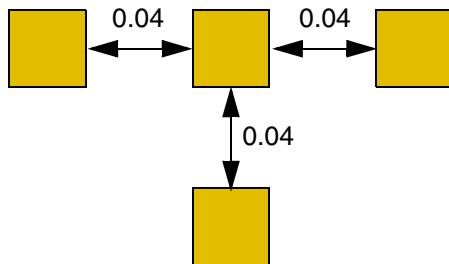
d) The constraint does not apply because all four adjacent via cuts are on opposite corners.

**Example 10: viaSpacing with exceptWithin, numCuts, and distance**

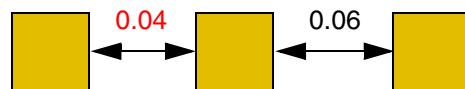
The edge-to-edge spacing between adjacent via cuts must be at least 0.06 if a via cut has two or more neighboring via cuts at a distance less than 0.07 away. The constraint does not apply if all adjacent via cuts are at a distance less than 0.05.

```
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).

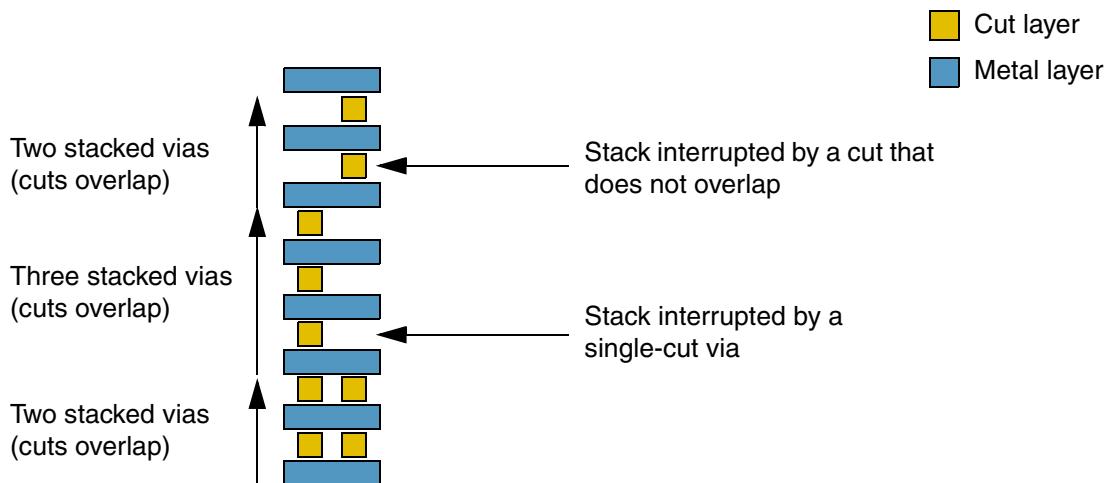
## **viaStackingLimits**

```
viaStackingLimits(  
    ( x_stackLimit  
        [ 'noSingleCut'  
        [ 'exceptIndividualArea f_individualArea 'exceptSumArea f_sumArea]  
        [ 'lowCutSizeWithin]  
        [ tx_bottomLayer tx_topLayer]  
    )  
)
```

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.

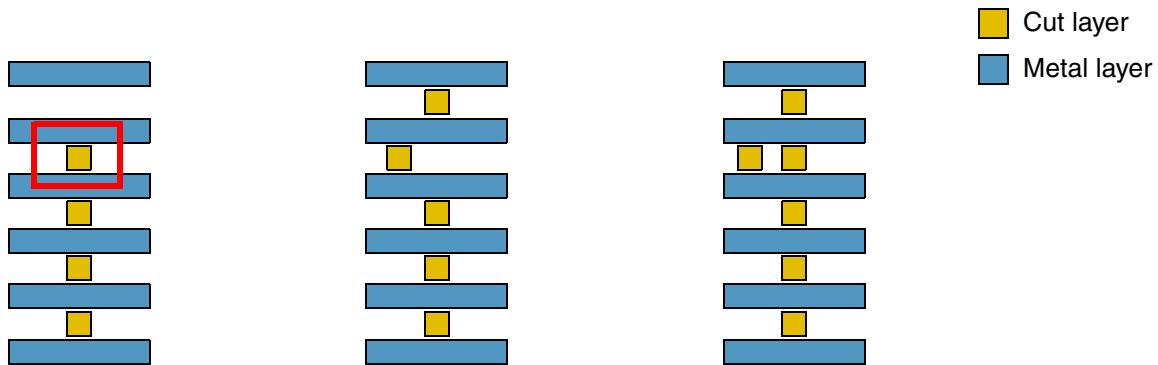
The figure below illustrates via stacks.



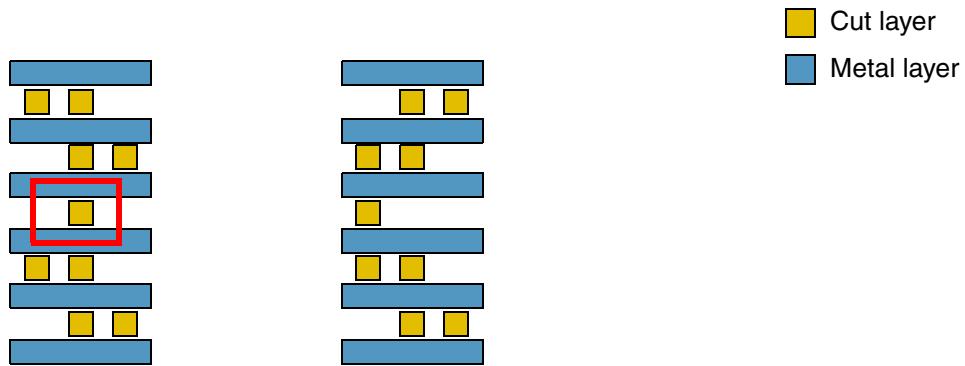
## Virtuoso Technology Data Constraint Reference

### 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.

## Parameters

*tx\_bottomLayer tx\_topLayer*

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.

Valid values: The layer name or the layer number

'noSingleCut

This parameter specifies whether stacked vias can be single-cut or 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.

'exceptIndividualArea

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.

'exceptSumArea

The constraint does not apply if the union sum of the metal shape containing a stacked via on layers other than the bottommost and topmost metal layers is greater than or equal to this value.

'lowCutSizeWithin

If in a via the upper adjacent cut is within the minimum cut size of the lower layer, it is considered to be a stacked via.

## Example

The via stack limit is set to 3 for Metal1 through Metal4 layers; 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 Constraint Reference**  
Via Construction Constraints

---

**Virtuoso Technology Data Constraint 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>

### Introduction

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.

## **Virtuoso Technology Data Constraint Reference**

### Constraint Name Mapping for “minEnclosure”, “minExtension”, and “maxEnclosure”

---

- 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 IC.6.1.4FCS and below. For example, the `minExtensionDistance` constraint cannot be loaded into OA by the IC.6.1.4FCS tech

## **Virtuoso Technology Data Constraint Reference**

Constraint Name Mapping for “minEnclosure”, “minExtension”, and “maxEnclosure”

---

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.

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 Constraint Reference**  
Constraint Name Mapping for “minEnclosure”, “minExtension”, and “maxEnclosure”

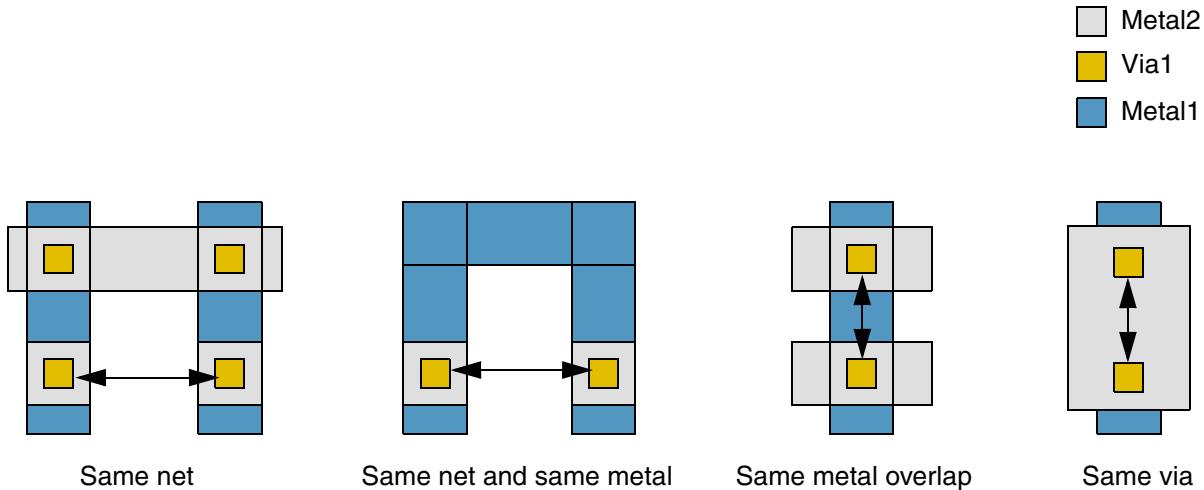
---

## Glossary

### Connectivity Type

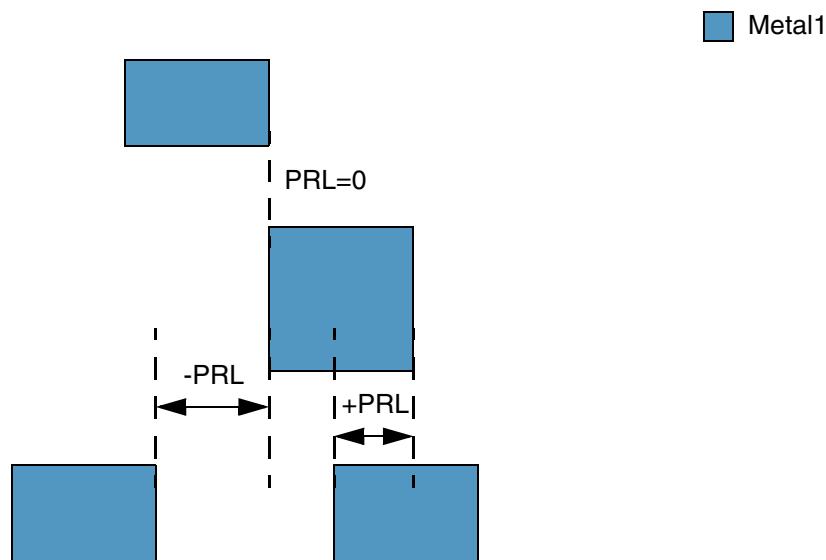
Specifies how shapes are connected. The various connectivity types are illustrated below using a pair of via cuts:

- 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:

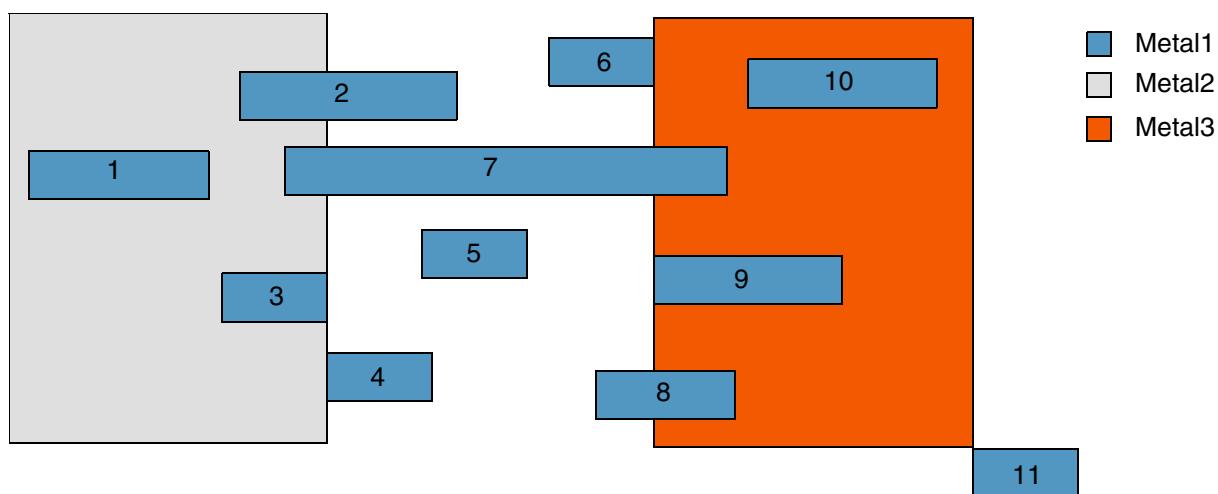
- ❑ "[*x\_lower* *x\_upper*]": The given value must be greater than or equal to *x\_lower* and less than or equal to *x\_upper*.
- ❑ "(*x\_lower* *x\_upper*)": The given value must be greater than *x\_lower* and less than *x\_upper*.
- ❑ "[*x\_lower* *x\_upper*)": The given value must be greater than 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\_valuex\_value*.
- ❑ "<=*x\_valuex\_value*.
- ❑ ">*x\_valuex\_value*.
- ❑ ">=*x\_valuex\_value*.
- ❑ *x\_value*: The given value must be equal to *x\_value*.

# Virtuoso Technology Data Constraint Reference

## Glossary

### 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.



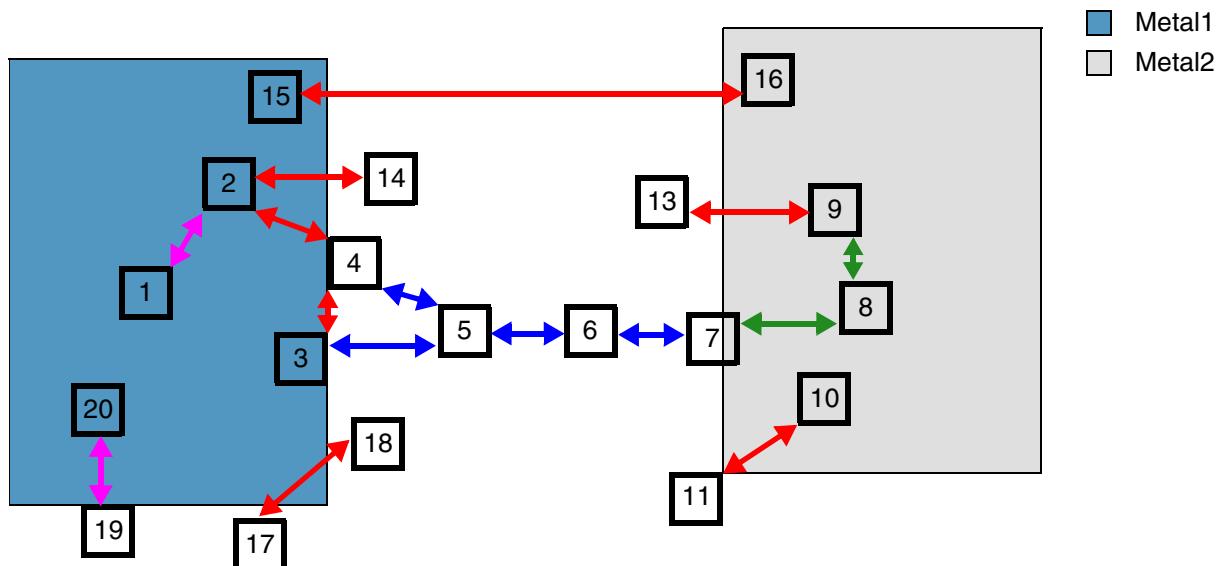
For all constraints, except area-specific enclosure constraints, multiple constraints are defined in an AND group. For example, if minArea is defined as shown below, the Metal2 'insideLayers constraint applies to shapes 1, 2, 3, and 7, and the Metal3 'insideLayers constraint applies to shapes 7, 8, 9, and 10, in the figure above. Both these constraints apply to shape 7. The Metal2 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.



## **Virtuoso Technology Data Constraint Reference**

### Glossary

---

#### **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 Constraint Reference**

### Glossary

---

---

## Loading "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.

**Note:** 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  $n^{\text{th}}$  entry is missing, the  $n+1^{\text{th}}$  entry is copied to the  $n^{\text{th}}$  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 Constraint Reference

### Loading "CDBA Type" Tables

---

**Example 1: A 2-D legacy table without a default value, which is considered as a "CDBA style" table**

```
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
          (0.035 0.145 ) 0.084
          (0.035 0.155 ) 0.084
        )
      )
    ) ;spacingTables
  ) ;foundry
) ;constraintGroups
```

## Virtuoso Technology Data Constraint Reference

### Loading "CDBA Type" Tables

---

```
(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
)
)
) ;spacingTables
) ;foundry
```

## Virtuoso Technology Data Constraint Reference

### Loading "CDBA Type" Tables

---

```
) ;constraintGroups
```

#### **Example 2: A table with a default value, which is considered as an "OA style" table**

```
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.

**Note:** We recommend that you move away from the CDBA-style tables.