



---

# **Standard Verification Rule Format (SVRF) Manual**

**Calibre® 2011.3**

---

**© 1996-2011 Mentor Graphics Corporation  
All rights reserved.**

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### **RESTRICTED RIGHTS LEGEND 03/97**

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

**Contractor/manufacturer is:**

Mentor Graphics Corporation

8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

Telephone: 503.685.7000

Toll-Free Telephone: 800.592.2210

Website: [www.mentor.com](http://www.mentor.com)

SupportNet: [supportnet.mentor.com/](http://supportnet.mentor.com/)

Send Feedback on Documentation: [supportnet.mentor.com/user/feedback\\_form.cfm](http://supportnet.mentor.com/user/feedback_form.cfm)

**TRADEMARKS:** The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other third parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the respective third-party owner. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: [www.mentor.com/terms\\_conditions/trademarks.cfm](http://www.mentor.com/terms_conditions/trademarks.cfm).

# Table of Contents

---

<b>Chapter 1</b>	
<b>Preface .....</b>	<b>35</b>
Audience .....	35
Documentation Conventions .....	36
<b>Chapter 2</b>	
<b>Key Concepts.....</b>	<b>39</b>
Rule File Overview .....	40
Compilation Overview .....	41
General Syntactic Conventions .....	41
SVRF Symbols and Command Parsing .....	41
Comments .....	44
Constraints .....	45
Numeric Expressions .....	46
Negative Numbers in Coordinate Lists.....	46
String Constants .....	46
Variables .....	47
Environment Variables in Pathname Parameters .....	47
Keyword Abbreviations .....	48
Keyword - Name Conflicts.....	49
Rules for Inlined Parameter Blocks .....	50
Basic Rule File Structure .....	52
Rule File Elements .....	52
Pre-Processor Directives.....	56
Macros .....	64
<b>Chapter 3</b>	
<b>Summary of Rule File Elements .....</b>	<b>69</b>
Layer Operations .....	69
Dimensional Check Layer Operations .....	70
Auxiliary Layer Operations .....	71
Connectivity Extraction Statements.....	80
Virtual Connect Specification Statements .....	81
Device Recognition Statement.....	82
Fracturing Statements .....	82
Post-Design Correction Statements .....	83
Specification Statements .....	84
DFM Specification Statements .....	84
nmDRC Specification Statements.....	85
ERC Specification Statements .....	88
Layout Database Specification Statements.....	89
LVS Specification Statements .....	94

PERC Specification Statements .....	104
PEX Specification Statements .....	104
PEX Inductance Specification Statements .....	110
Unit Specification Statements.....	112
Other Specification Statements.....	112
<b>Chapter 4</b>	
<b>Reference Dictionary .....</b>	<b>114</b>
AND .....	115
Angle.....	118
Area.....	120
Attach .....	121
Capacitance Order.....	124
Coincident Edge .....	126
Coincident Inside Edge.....	127
Coincident Outside Edge .....	128
Connect .....	129
Convex Edge.....	134
Copy .....	138
Cut.....	141
DBCLASSIFY .....	144
Deangle.....	154
Density .....	156
Density Convolve.....	183
Device .....	196
Device Layer.....	216
DFM Analyze .....	220
DFM Assert.....	228
DFM CAF.....	232
DFM Classify .....	237
DFM Compress.....	240
DFM Connectivity Redundant .....	243
DFM Copy .....	248
DFM Create Layer .....	252
DFM Critical Area .....	254
DFM Database .....	268
DFM Defaults .....	272
DFM Expand Edge .....	275
DFM Expand Enclosure .....	280
DFM Fill .....	285
DFM Function.....	289
DFM GCA .....	301
DFM Grow .....	304
DFM Histogram .....	307
DFM Measure.....	311
DFM NARAC.....	319
DFM Property .....	321
DFM Property Merge .....	333

## Table of Contents

---

DFM RDB . . . . .	335
DFM Read . . . . .	346
DFM Redundant Vias . . . . .	349
DFM Select Check . . . . .	352
DFM Shift Edge . . . . .	354
DFM Size . . . . .	357
DFM Space . . . . .	362
DFM Spec Fill . . . . .	365
DFM Spec Fill Optimizer . . . . .	390
DFM Spec Fill Shape . . . . .	397
DFM Spec Spatial Sample . . . . .	404
DFM Spec Via Redundancy . . . . .	406
DFM Stamp . . . . .	408
DFM Summary Report . . . . .	409
DFM Text . . . . .	411
DFM Transform . . . . .	413
DFM Transition . . . . .	415
DFM Unselect Check . . . . .	424
DFM YS Autostart . . . . .	425
Disconnect . . . . .	426
Donut . . . . .	428
Drawn Acute . . . . .	430
Drawn Angled . . . . .	431
Drawn Offgrid . . . . .	432
Drawn Skew . . . . .	433
DRC Boolean Nosnap45 . . . . .	434
DRC Cell Name . . . . .	436
DRC Cell Text . . . . .	439
DRC Check Map . . . . .	440
DRC Check Text . . . . .	453
DRC Exclude False Notch . . . . .	454
DRC Incremental Connect . . . . .	455
DRC Incremental Connect Warning . . . . .	459
DRC Keep Empty . . . . .	461
DRC Magnify Density . . . . .	462
DRC Magnify NAR . . . . .	463
DRC Magnify Results . . . . .	464
DRC Map Text . . . . .	468
DRC Map Text Depth . . . . .	470
DRC Map Text Layer . . . . .	471
DRC Maximum Cell Name Length . . . . .	473
DRC Maximum Results . . . . .	474
DRC Maximum Unattached Label Warnings . . . . .	475
DRC Maximum Vertex . . . . .	476
DRC Print Area . . . . .	478
DRC Print Perimeter . . . . .	479
DRC Results Database . . . . .	480
DRC Results Database Libname . . . . .	488
DRC Results Database Precision . . . . .	489

---

## Table of Contents

DRC Select Check . . . . .	491
DRC Select Check By Layer . . . . .	492
DRC Summary Report . . . . .	494
DRC Tolerance Factor . . . . .	496
DRC Tolerance Factor NAR . . . . .	497
DRC Unselect Check . . . . .	499
DRC Unselect Check By Layer . . . . .	500
Enclose . . . . .	502
Enclose Rectangle . . . . .	505
Enclosure . . . . .	506
ERC Cell Name . . . . .	530
ERC Check Text . . . . .	532
ERC Keep Empty . . . . .	533
ERC Maximum Results . . . . .	534
ERC Maximum Vertex . . . . .	535
ERC Path Also . . . . .	536
ERC Pathchk . . . . .	537
ERC Results Database . . . . .	544
ERC Select Check . . . . .	546
ERC Summary Report . . . . .	549
ERC Unselect Check . . . . .	551
Exclude Acute . . . . .	552
Exclude Angled . . . . .	553
Exclude Cell . . . . .	554
Exclude Offgrid . . . . .	555
Exclude Skew . . . . .	556
Expand Cell . . . . .	557
Expand Cell Text . . . . .	558
Expand Edge . . . . .	560
Expand Text . . . . .	564
Extent . . . . .	565
Extent Cell . . . . .	567
Extent Drawn . . . . .	571
Extents . . . . .	574
External . . . . .	576
Flag Acute . . . . .	601
Flag Angled . . . . .	603
Flag Nonsimple . . . . .	605
Flag Nonsimple Path . . . . .	607
Flag Offgrid . . . . .	609
Flag Skew . . . . .	611
Flatten . . . . .	613
Flatten Cell . . . . .	614
Flatten Inside Cell . . . . .	615
Fracture HITACHI . . . . .	616
Fracture JEOL . . . . .	618
Fracture MEBES . . . . .	620
Fracture MICRONIC . . . . .	622
Fracture NUFLARE . . . . .	624

## Table of Contents

---

Fracture VBOASIS . . . . .	627
Fracture OASIS_MASK . . . . .	629
Group . . . . .	631
Grow . . . . .	632
Hcell . . . . .	637
Holes . . . . .	640
Include . . . . .	644
Inductance MICheck . . . . .	646
Inductance Wire . . . . .	649
Inside . . . . .	650
Inside Cell . . . . .	652
Inside Edge . . . . .	656
Interact . . . . .	658
Internal . . . . .	662
Label Order . . . . .	683
Layer . . . . .	685
Layer Directory . . . . .	689
Layer Map . . . . .	691
Layer Resolution . . . . .	696
Layout Allow Duplicate Cell . . . . .	697
Layout Base Cell . . . . .	698
Layout Base Layer . . . . .	699
Layout Bump2 . . . . .	701
Layout Case . . . . .	703
Layout Cell List . . . . .	705
Layout Cell Match Rule . . . . .	709
Layout Clone By Layer . . . . .	712
Layout Clone Rotated Placements . . . . .	714
Layout Clone Transformed Placements . . . . .	716
Layout Depth . . . . .	719
Layout Error On Input . . . . .	720
Layout Ignore Text . . . . .	722
Layout Input Exception RDB . . . . .	725
Layout Input Exception Severity . . . . .	729
Layout Magnify . . . . .	750
Layout Merge On Input . . . . .	753
Layout Path . . . . .	755
Layout Path2 . . . . .	761
Layout Place Cell . . . . .	765
Layout Polygon . . . . .	767
Layout Precision . . . . .	769
Layout Preserve Case . . . . .	771
Layout Preserve Cell List . . . . .	774
Layout Primary . . . . .	775
Layout Primary2 . . . . .	777
Layout Process Box Record . . . . .	779
Layout Process Node Record . . . . .	780
Layout Property Audit . . . . .	781
Layout Property Text . . . . .	783

Layout Property Text OASIS . . . . .	784
Layout Rename Cell . . . . .	785
Layout Rename ICV . . . . .	786
Layout Rename Text . . . . .	787
Layout System . . . . .	796
Layout System2 . . . . .	802
Layout Text . . . . .	804
Layout Text File . . . . .	806
Layout Top Layer . . . . .	808
Layout Use Database Precision . . . . .	810
Layout Windel . . . . .	811
Layout Windel Cell . . . . .	813
Layout Windel Layer . . . . .	815
Layout Window . . . . .	816
Layout Window Cell . . . . .	818
Layout Window Clip . . . . .	820
Layout Window Layer . . . . .	821
Length . . . . .	823
Litho DenseOPC . . . . .	824
Litho File . . . . .	826
Litho OPC . . . . .	827
Litho OPCverify . . . . .	829
Litho ORC . . . . .	831
Litho Printimage . . . . .	833
Litho PSMgate . . . . .	835
LVS Abort On ERC Error . . . . .	836
LVS Abort On Softchk . . . . .	837
LVS Abort On Supply Error . . . . .	838
LVS All Capacitor Pins Swappable . . . . .	840
LVS Annotate Devices . . . . .	841
LVS Auto Expand Hcells . . . . .	843
LVS Box . . . . .	846
LVS Builtin Device Pin Swap . . . . .	850
LVS Builtin MOS NRD_NRS . . . . .	852
LVS Cell List . . . . .	854
LVS Cell Supply . . . . .	858
LVS Center Device Pins . . . . .	860
LVS Check Port Names . . . . .	863
LVS Compare Case . . . . .	865
LVS Component Subtype Property . . . . .	868
LVS Component Type Property . . . . .	869
LVS Cpoint . . . . .	870
LVS Device Type . . . . .	874
LVS Discard Pins By Device . . . . .	880
LVS Downcase Device . . . . .	883
LVS EDDM Process M . . . . .	885
LVS Exact Subtypes . . . . .	888
LVS Exclude Hcell . . . . .	890
LVS Execute ERC . . . . .	891

## Table of Contents

---

LVS Expand Seed Promotions . . . . .	892
LVS Expand Unbalanced Cells . . . . .	893
LVS Filter . . . . .	895
LVS Filter Unused Bipolar . . . . .	900
LVS Filter Unused Capacitors . . . . .	901
LVS Filter Unused Diodes . . . . .	902
LVS Filter Unused MOS . . . . .	903
LVS Filter Unused Option . . . . .	904
LVS Filter Unused Resistors . . . . .	910
LVS Flatten Inside Cell . . . . .	911
LVS Global Layout Name . . . . .	914
LVS Globals Are Ports . . . . .	917
LVS Ground Name . . . . .	918
LVS Heap Directory . . . . .	920
LVS Ignore Ports . . . . .	922
LVS Ignore Trivial Named Ports . . . . .	923
LVS Inject Logic . . . . .	925
LVS Isolate Shorts . . . . .	927
LVS Map Device . . . . .	938
LVS MOS Swappable Properties . . . . .	941
LVS Netlist All Texted Pins . . . . .	942
LVS Netlist Allow Inconsistent Model . . . . .	943
LVS Netlist Box Contents . . . . .	945
LVS Netlist Comment Coded Properties . . . . .	946
LVS Netlist Comment Coded Substrate . . . . .	948
LVS Netlist Unnamed Box Pins . . . . .	949
LVS NL Pin Locations . . . . .	950
LVS Non User Name . . . . .	951
LVS Out Of Range Exclude Zero . . . . .	954
LVS Pin Name Property . . . . .	956
LVS Power Name . . . . .	957
LVS Precise Interaction . . . . .	959
LVS Preserve Box Cells . . . . .	960
LVS Preserve Box Ports . . . . .	962
LVS Preserve Floating Top Nets . . . . .	964
LVS Preserve Parameterized Cells . . . . .	965
LVS Property Initialize . . . . .	966
LVS Property Map . . . . .	968
LVS Property Resolution Maximum . . . . .	971
LVS Push Devices . . . . .	972
LVS Recognize Gates . . . . .	986
LVS Recognize Gates Tolerance . . . . .	990
LVS Reduce . . . . .	995
LVS Reduce Parallel Bipolar . . . . .	1004
LVS Reduce Parallel Capacitors . . . . .	1007
LVS Reduce Parallel Diodes . . . . .	1010
LVS Reduce Parallel MOS . . . . .	1013
LVS Reduce Parallel Resistors . . . . .	1016
LVS Reduce Semi Series MOS . . . . .	1019

LVS Reduce Series Capacitors . . . . .	1020
LVS Reduce Series MOS . . . . .	1023
LVS Reduce Series Resistors . . . . .	1026
LVS Reduce Split Gates . . . . .	1029
LVS Reduction Priority . . . . .	1033
LVS Report . . . . .	1035
LVS Report Maximum . . . . .	1036
LVS Report Option . . . . .	1037
LVS Report Units . . . . .	1048
LVS Report Warnings Hcell Only . . . . .	1050
LVS Report Warnings Top Only . . . . .	1051
LVS Reverse WL . . . . .	1052
LVS Short Equivalent Nodes . . . . .	1053
LVS Show Seed Promotions . . . . .	1055
LVS Show Seed Promotions Maximum . . . . .	1059
LVS Signature Maximum . . . . .	1060
LVS Soft Substrate Pins . . . . .	1061
LVS Softchk . . . . .	1062
LVS Spice Allow Duplicate Subcircuit Names . . . . .	1065
LVS Spice Allow Floating Pins . . . . .	1066
LVS Spice Allow Inline Parameters . . . . .	1067
LVS Spice Allow Unquoted Strings . . . . .	1070
LVS Spice Conditional LDD . . . . .	1071
LVS Spice Cull Primitive Subcircuits . . . . .	1072
LVS Spice Implied MOS Area . . . . .	1074
LVS Spice Multiplier Name . . . . .	1075
LVS Spice Option . . . . .	1077
LVS Spice Override Globals . . . . .	1079
LVS Spice Prefer Pins . . . . .	1082
LVS Spice Redefine Param . . . . .	1083
LVS Spice Rename Parameter . . . . .	1085
LVS Spice Replicate Devices . . . . .	1091
LVS Spice Scale X Parameters . . . . .	1094
LVS Spice Slash Is Space . . . . .	1096
LVS Spice Strict WL . . . . .	1097
LVS Split Gate Ratio . . . . .	1099
LVS Strict Subtypes . . . . .	1105
LVS Summary Report . . . . .	1106
LVS Write Injected Layout Netlist . . . . .	1107
LVS Write Injected Source Netlist . . . . .	1109
LVS Write Layout Netlist . . . . .	1111
LVS Write Source Netlist . . . . .	1113
Magnify . . . . .	1115
Mask Results Database . . . . .	1116
Mask SVDB Directory . . . . .	1118
MDP Checkmap . . . . .	1128
MDP Embed . . . . .	1130
MDP Mapsize . . . . .	1132
MDP Maskopt . . . . .	1137

## Table of Contents

---

MDP Oasis_Extent . . . . .	1140
MDPmerge . . . . .	1143
MDPstat . . . . .	1144
MDPverify . . . . .	1145
Merge . . . . .	1147
Net . . . . .	1149
Net Area . . . . .	1151
Net Area Ratio . . . . .	1152
Net Area Ratio Accumulate . . . . .	1173
Net Area Ratio Print . . . . .	1177
Net Interact . . . . .	1178
NOT . . . . .	1180
Not Angle . . . . .	1181
Not Area . . . . .	1183
Not Coincident Edge . . . . .	1184
Not Coincident Inside Edge . . . . .	1185
Not Coincident Outside Edge . . . . .	1186
Not Cut . . . . .	1187
Not Donut . . . . .	1189
Not Enclose . . . . .	1190
Not Enclose Rectangle . . . . .	1192
Not Inside . . . . .	1193
Not Inside Cell . . . . .	1195
Not Inside Edge . . . . .	1198
Not Interact . . . . .	1199
Not Length . . . . .	1202
Not Net . . . . .	1203
Not Outside . . . . .	1204
Not Outside Edge . . . . .	1205
Not Rectangle . . . . .	1206
Not Touch . . . . .	1209
Not Touch Edge . . . . .	1211
Not Touch Inside Edge . . . . .	1213
Not Touch Outside Edge . . . . .	1215
Not With Edge . . . . .	1217
Not With Neighbor . . . . .	1218
Not With Text . . . . .	1223
Not With Width . . . . .	1224
Offgrid . . . . .	1225
Opcbias . . . . .	1235
Opclineend . . . . .	1245
Opcsbar . . . . .	1255
OR . . . . .	1281
OR Edge . . . . .	1283
Ornet . . . . .	1285
Outside . . . . .	1286
Outside Edge . . . . .	1287
Parasitic Variation . . . . .	1288
Path Length . . . . .	1291

Pathchk . . . . .	1293
PERC Load . . . . .	1304
PERC Netlist . . . . .	1308
PERC Pattern Path . . . . .	1310
PERC Property . . . . .	1312
PERC Report . . . . .	1314
PERC Report Maximum . . . . .	1315
PERC Report Option . . . . .	1316
PERC Report Placement List Maximum . . . . .	1317
PERC Waiver Path . . . . .	1318
Perimeter . . . . .	1319
PEX 3DReference . . . . .	1320
PEX Alias . . . . .	1322
PEX BA Mapfile . . . . .	1327
PEX Bulk Layer . . . . .	1335
PEX CMP Mode . . . . .	1339
PEX Contact Capacitance . . . . .	1341
PEX Corner . . . . .	1342
PEX Corner Custom . . . . .	1343
PEX DEF Extract Cell Obstructions . . . . .	1346
PEX DEF Map . . . . .	1347
PEX Density Estimate . . . . .	1348
PEX Density Window . . . . .	1349
PEX Driver File . . . . .	1351
PEX Elayer . . . . .	1352
PEX Exclude Distributed . . . . .	1353
PEX Exclude Lumped . . . . .	1355
PEX Extract Exclude . . . . .	1357
PEX Extract Floating Nets . . . . .	1359
PEX Extract Include . . . . .	1362
PEX Extract Rgate . . . . .	1364
PEX Extract Temperature . . . . .	1369
PEX Fieldsolver Boundary . . . . .	1371
PEX Fieldsolver Cell_array . . . . .	1376
PEX Fieldsolver Endcap Spacing . . . . .	1385
PEX Fieldsolver Mode . . . . .	1386
PEX Fill Handling . . . . .	1389
PEX Fill Model . . . . .	1390
PEX Fracture Frequency . . . . .	1391
PEX Generate Driver File Tag . . . . .	1392
PEX Generate Driver_File Tag . . . . .	1393
PEX Ground . . . . .	1394
PEX Ground Layer . . . . .	1395
PEX Ideal Xcell . . . . .	1396
PEX Ignore Capacitance . . . . .	1398
PEX Ignore Inductance . . . . .	1410
PEX Ignore Resistance . . . . .	1411
PEX Include Distributed . . . . .	1413
PEX Include Lumped . . . . .	1415

## Table of Contents

---

PEX Indie Spacing . . . . .	1417
PEX Inductance Default Partial Model. . . . .	1418
PEX Inductance Default PI. . . . .	1419
PEX Inductance Differential Pair . . . . .	1420
PEX Inductance Doprocess. . . . .	1422
PEX Inductance Driver File . . . . .	1423
PEX Inductance Driver Summary . . . . .	1424
PEX Inductance Extract Layers . . . . .	1425
PEX Inductance Filter. . . . .	1426
PEX Inductance Forward Coupling . . . . .	1429
PEX Inductance Frequency. . . . .	1430
PEX Inductance ... Frequency. . . . .	1431
PEX Inductance Layer Summary . . . . .	1433
PEX Inductance MICHECK Constraint . . . . .	1434
PEX Inductance Minlength. . . . .	1435
PEX Inductance Parameters . . . . .	1436
PEX Inductance Range . . . . .	1437
PEX Inductance Returnpath . . . . .	1438
PEX Inductance Same Net Mutual . . . . .	1440
PEX Inductance Self. . . . .	1441
PEX Inductance Skin Include . . . . .	1442
PEX Inductance Switch Time. . . . .	1443
PEX Inductance Switch_Time . . . . .	1444
PEX Inductance ... Switch Time. . . . .	1445
PEX Inductance ... Switch_Time . . . . .	1447
PEX Inductance Victim . . . . .	1448
PEX Inductance Victim Path . . . . .	1450
PEX Inductance Victim_Path . . . . .	1452
PEX Inductance Victim File . . . . .	1453
PEX Inductance Victim_File . . . . .	1454
PEX Magnify . . . . .	1455
PEX Map. . . . .	1456
PEX Netlist . . . . .	1458
PEX Netlist ADMS. . . . .	1472
PEX Netlist Capacitance Unit. . . . .	1475
PEX Netlist Character Map . . . . .	1476
PEX Netlist Connection Section. . . . .	1478
PEX Netlist Create Smashed Device Names . . . . .	1479
PEX Netlist Device Resistance Model . . . . .	1480
PEX Netlist Distributed . . . . .	1481
PEX Netlist Escape Characters. . . . .	1487
PEX Netlist Export Ports . . . . .	1488
PEX Netlist Filter . . . . .	1489
PEX Netlist Global Nets . . . . .	1491
PEX Netlist Linewrap. . . . .	1492
PEX Netlist LPE Ignore Idealnet . . . . .	1493
PEX Netlist LPE Using Extmode . . . . .	1494
PEX Netlist Lumped. . . . .	1495
PEX Netlist Mutual Resistance. . . . .	1498

PEX Netlist Noxref Net Names . . . . .	1499
PEX Netlist Position File . . . . .	1500
PEX Netlist Replicated_Device Delimiter . . . . .	1501
PEX Netlist SchematicOnly . . . . .	1503
PEX Netlist Select File . . . . .	1508
PEX Netlist Simple . . . . .	1515
PEX Netlist Smashed_Device Delimiter . . . . .	1517
PEX Netlist Subnode Section . . . . .	1518
PEX Netlist Unshort Device Pins . . . . .	1519
PEX Netlist Uppercase Keywords . . . . .	1520
PEX Netlist Virtual Connect . . . . .	1521
PEX Pin Order . . . . .	1522
PEX Power . . . . .	1524
PEX Probe File . . . . .	1525
PEX Reduce Analog . . . . .	1527
PEX Reduce CC . . . . .	1529
PEX Reduce Digital . . . . .	1531
PEX Reduce Distributed . . . . .	1533
PEX Reduce Mincap . . . . .	1534
PEX Reduce Minres . . . . .	1536
PEX Reduce ROnly . . . . .	1538
PEX Reduce TICER . . . . .	1540
PEX Reduce Via Resistance . . . . .	1544
PEX Report . . . . .	1548
PEX Report Coupling Capacitance . . . . .	1549
PEX Report Distributed . . . . .	1551
PEX Report Lumped . . . . .	1553
PEX Report Mutual Inductance . . . . .	1554
PEX Report Netsummary . . . . .	1556
PEX Report Point2Point . . . . .	1559
PEX Resistance Parameters . . . . .	1563
PEX Sensitivity . . . . .	1570
PEX Skin Include . . . . .	1571
PEX Slots Handling . . . . .	1572
PEX Temperature . . . . .	1574
PEX Thickness EQN . . . . .	1575
PEX Thickness Nominal . . . . .	1577
PEX Threshold . . . . .	1578
PEX Tolerance Distributed . . . . .	1580
PEX Via Capacitance . . . . .	1582
PEX Via Reduction Resistance . . . . .	1583
PEX Xcell . . . . .	1584
PEX Xcell Extract Mode . . . . .	1586
PEX Xcell Precedence . . . . .	1588
Pins . . . . .	1591
Polygon . . . . .	1592
Port Depth . . . . .	1594
Port Layer Polygon . . . . .	1596
Port Layer Text . . . . .	1599

## Table of Contents

---

Ports . . . . .	1601
Precision . . . . .	1602
Push. . . . .	1605
Push Cell. . . . .	1607
Rectangle. . . . .	1608
Rectangle Enclosure . . . . .	1611
Rectangles. . . . .	1618
Resistance Connection . . . . .	1625
Resistance Device_Seed . . . . .	1628
Resistance Rho . . . . .	1629
Resistance Sheet . . . . .	1631
Resolution . . . . .	1633
RET NMDPC . . . . .	1634
RET PXOPC . . . . .	1636
RET SBAR . . . . .	1639
RET Sraf_Fill . . . . .	1641
Rotate . . . . .	1643
Sconnect . . . . .	1644
Shift. . . . .	1651
Shrink . . . . .	1653
Size . . . . .	1656
Snap. . . . .	1664
Snap Offgrid . . . . .	1665
Source Case. . . . .	1666
Source Path . . . . .	1668
Source Primary . . . . .	1670
Source System. . . . .	1671
Stamp . . . . .	1672
SVRF Error . . . . .	1675
SVRF Message . . . . .	1676
SVRF Version. . . . .	1677
TDDRC. . . . .	1678
Text. . . . .	1685
Text Depth . . . . .	1687
Text Layer. . . . .	1689
Text Print Maximum. . . . .	1690
Title. . . . .	1691
Topex . . . . .	1692
Touch . . . . .	1693
Touch Edge. . . . .	1696
Touch Inside Edge . . . . .	1698
Touch Outside Edge . . . . .	1700
Trace Property. . . . .	1702
Unit Capacitance. . . . .	1718
Unit Inductance . . . . .	1720
Unit Length. . . . .	1722
Unit Resistance . . . . .	1724
Unit Time . . . . .	1726
Variable. . . . .	1727

Vertex .....	1732
Virtual Connect Box Colon .....	1733
Virtual Connect Box Name .....	1735
Virtual Connect Colon .....	1737
Virtual Connect Depth .....	1740
Virtual Connect Incremental .....	1742
Virtual Connect Name .....	1744
Virtual Connect Report .....	1746
Virtual Connect Report Maximum .....	1753
Virtual Connect Semicolon As Colon .....	1754
With Edge .....	1755
With Neighbor .....	1757
With Text .....	1762
With Width .....	1763
XOR .....	1768
<b>Chapter 5</b>	
<b>Built-In Languages .....</b>	<b>1771</b>
Common Features of SVRF Built-In Languages .....	1771
Numeric Functions for Built-in Languages .....	1775
Device Property Computation Built-In Language .....	1776
Writing a Device Property Computation Program .....	1777
Property Computation Language Characteristics .....	1779
Data Sources .....	1780
DEBUG Statement .....	1781
PROPerty Statement .....	1781
Summary of Device Property Computation Functions .....	1782
Pin and Layer References .....	1786
Function Reference .....	1786
TVF Functional Interface for Device Property Calculation .....	1827
Well Proximity Calculation Methods and Examples .....	1833
Device Annotation Built-In Language .....	1842
Writing a Device Annotation Program .....	1842
Device Annotation Language Characteristics .....	1843
Data Sources .....	1844
DEBUG Statement .....	1845
SELECT Statement .....	1845
PROPerty Statement .....	1846
Summary of Device Annotation Functions .....	1847
Using DFM Property with Device Layer ANNOTATE Programs .....	1848
Device Reduction Effective Property Computation Language .....	1851
Writing an Effective Property Computation Program .....	1851
Effective Property Language Characteristics .....	1855
Data Sources .....	1856
EFFECTIVE and EFFECTIVE STRING Statements .....	1856
Limitations on String Properties .....	1857
DEBUG Statement .....	1858
WARN Statement .....	1858

## Table of Contents

---

Effective Property Language Vector Functions . . . . .	1859
Default Reduction of String Properties . . . . .	1862
Effective Property Computation Examples . . . . .	1862
Effective Property Computation Limitations . . . . .	1864
Reduction Program Semantic Checking . . . . .	1864
LVS Property Initialize Built-In Language . . . . .	1867
LVS Property Initialize Characteristics . . . . .	1868
Data Sources . . . . .	1869
PROProperty Statement . . . . .	1869
DEBUG Statement . . . . .	1870
WARN Statement . . . . .	1870
Property Initialization Functions . . . . .	1871
Trace Property Computation and Reporting Language . . . . .	1872
Writing a Trace Property Program . . . . .	1872
Trace Property Language Characteristics . . . . .	1875
Data Sources . . . . .	1876
PROProperty and PROProperty STRING Statements . . . . .	1877
DEBUG Statement . . . . .	1878
WARN Statement . . . . .	1879
Trace Property Function Summary . . . . .	1879
Trace Property Function Reference . . . . .	1880
Trace Property User Computation Example . . . . .	1883
Trace Property Program Limitations . . . . .	1884
<b>Chapter 6</b>	
<b>Tcl Verification Format . . . . .</b>	<b>1889</b>
Introduction . . . . .	1889
Differences Between TVF and SVRF . . . . .	1890
Compile-Time TVF . . . . .	1891
Compile-Time TVF Commands . . . . .	1892
Opening Line . . . . .	1892
Command Syntax . . . . .	1892
Passing Verbatim SVRF Code in Compile-Time TVF . . . . .	1893
SVRF Statements Mapped to TVF . . . . .	1893
SVRF Statements Not Mapped to TVF . . . . .	1896
Comment Characters . . . . .	1896
Layer Definitions and Assignments . . . . .	1896
Rule Checks . . . . .	1898
Handling of Special Characters . . . . .	1901
Passing Command Line Arguments in Compile-Time TVF . . . . .	1902
Compile-Time TVF File Usage . . . . .	1903
Identifying Errors in Compile-Time TVF . . . . .	1904
Runtime TVF . . . . .	1906
TVF Function . . . . .	1906
TVF Layer Operation . . . . .	1908
Coding Guidelines . . . . .	1909
Layers in Runtime TVF . . . . .	1911
Runtime TVF Commands . . . . .	1911

Layer Functions . . . . .	1911
Layer Operations . . . . .	1913
Using Variables . . . . .	1914
System Variables . . . . .	1915
Comment Character . . . . .	1916
Examples . . . . .	1917
Echoing Layer Operation Calls to stdout . . . . .	1919
Error Reporting . . . . .	1919
Namespace Importation . . . . .	1921
Precautions for the Compile-Time Environment . . . . .	1922
TVF Command Reference . . . . .	1924
Global Variables . . . . .	1931
TVF Examples . . . . .	1932
<b>Chapter 7</b>	
<b>Error Messages . . . . .</b>	<b>1937</b>
Compilation Errors . . . . .	1937
Continuous nmDRC Errors . . . . .	1938
LVS Cell List Statement Errors . . . . .	1938
Constraint Errors . . . . .	1938
DRC Check Map Statement Errors . . . . .	1940
Pre-Processor Directive Errors . . . . .	1943
Density Operation Errors . . . . .	1943
Device Definition Errors . . . . .	1945
DFM Operation Errors . . . . .	1948
Device Property Specification Errors . . . . .	1959
LVS Device Type Statement Errors . . . . .	1963
Device Layer Operation Errors . . . . .	1963
Environment Variable Errors . . . . .	1964
ERC Errors . . . . .	1964
Connectivity Extraction Errors . . . . .	1964
LVS Filter Statement Errors . . . . .	1966
Rule Check Group Errors . . . . .	1967
Included Rule File Errors . . . . .	1967
Operation Input Errors . . . . .	1967
Secondary Keyword Errors . . . . .	1971
LVS Annotate Devices Errors . . . . .	1973
Layer Definition Errors . . . . .	1973
Layout Cell List Statement Errors . . . . .	1974
Layout Cell Match Rule Errors . . . . .	1974
Layout MOS Swappable Properties Errors . . . . .	1974
Location Errors . . . . .	1975
LVS Push Devices Statement Errors . . . . .	1975
LVS Property Initialize Statement Errors . . . . .	1975
PERC Load Statement Errors . . . . .	1977
LVS Property Map Statement Errors . . . . .	1977
PERC Property Statement Errors . . . . .	1979
LVS Reduce Statement Errors . . . . .	1979

## Table of Contents

---

LVS Short Equivalent Nodes Statement Errors . . . . .	1981
Litho Operation Errors . . . . .	1981
Macro Errors . . . . .	1983
LVS Map Device Statement Errors . . . . .	1983
Net Area Ratio Errors . . . . .	1985
Numeric Errors . . . . .	1987
Layer Statement Errors . . . . .	1989
OPC Operation Errors . . . . .	1989
General File Input Output Errors . . . . .	1992
Rule Check Statement Errors . . . . .	1992
Parasitic Extraction Errors . . . . .	1992
PEX Statement Errors . . . . .	1996
PEX Property Computation Errors . . . . .	1997
RDB Conflict Errors . . . . .	1999
Layer and Variable Resolution Errors . . . . .	1999
LVS Recognize Gates Tolerance Statement Errors . . . . .	2001
LVS Split Gate Ratio Statement Errors . . . . .	2001
General Specification Statement Errors . . . . .	2003
LVS Spice Rename Parameter Statement Errors . . . . .	2005
Syntax Errors . . . . .	2005
Trace Property Statement Errors . . . . .	2006
Layer Type Errors . . . . .	2008
SVRF Error Statement . . . . .	2008
Variable Statement Errors . . . . .	2008
SVRF Version Statement Errors . . . . .	2009
Runtime Messages . . . . .	2010
Calibre nmDRC-H Warnings . . . . .	2011
Calibre File Input Output Errors . . . . .	2013
Calibre Precision Exceptions . . . . .	2013
Calibre Cell and Placement Exceptions . . . . .	2015
Calibre Layout Input Errors . . . . .	2017
Calibre Layout Input Exceptions and Warnings . . . . .	2027
Calibre GDSII and OASIS Output Warnings . . . . .	2034
Calibre Output Warnings . . . . .	2036
Cell Name and Location Warnings . . . . .	2036
Connectivity Extraction and Stamp Warnings . . . . .	2038
DFM Warnings . . . . .	2040
ICrules Input Warnings . . . . .	2042
ICrules Notes . . . . .	2042
LVS Connectivity Extraction Messages . . . . .	2044
LVS Comparison Messages . . . . .	2048
LVS Stamp Error Messages . . . . .	2049
LVS Netlist Compilation Errors and Warnings . . . . .	2049
PERC TVF Errors . . . . .	2050
Maximum Results and Polygon Segmentation Warnings . . . . .	2056
Miscellaneous Warnings . . . . .	2056
Output File Open Warnings . . . . .	2058
Polygon Flagging Warnings . . . . .	2060
Rule File Errors . . . . .	2062

Unsupported Functionality Errors and Warnings . . . . .	2063
<b>Chapter 8</b>	
<b>Rule File Examples . . . . .</b>	<b>2069</b>
Sample Rule File . . . . .	2069
Width Checks. . . . .	2073
Spacing Checks . . . . .	2076
Enclosure and Extension Checks. . . . .	2089
Contact Checks . . . . .	2096
Deriving Layers . . . . .	2105
Other nmDRC Checks . . . . .	2109
Device Property Calculation Examples . . . . .	2113
ICVerify Applications. . . . .	2115

**Index****Third-Party Information****End-User License Agreement**

# List of Figures

---

Figure 4-1. One-layer Boolean AND Operation .....	116
Figure 4-2. Two-layer Boolean AND Operation.....	117
Figure 4-3. Angle Magnitudes in Relation to the X-axis.....	118
Figure 4-4. Area Operation .....	120
Figure 4-5. Attach Operation: .....	121
Figure 4-6. Multiple Attach Operations.....	122
Figure 4-7. Process Stack for Capacitance Order Example.....	125
Figure 4-8. Coincident Edge .....	126
Figure 4-9. Coincident Inside Edge .....	127
Figure 4-10. Coincident Outside Edge .....	128
Figure 4-11. Connect .....	130
Figure 4-12. Connect Operation Containing a Contact Layer.....	131
Figure 4-13. Convex Edge Simple Specification.....	135
Figure 4-14. Cut .....	142
Figure 4-15. Cut BY NET .....	143
Figure 4-16. Deangle Output Using 45-Degree Edges .....	155
Figure 4-17. Deangle Output Using Orthogonal Edges.....	155
Figure 4-18. Density WINDOW .....	159
Figure 4-19. Density WINDOW STEP .....	160
Figure 4-20. Density WINDOW TRUNCATE .....	161
Figure 4-21. Density WINDOW BACKUP .....	161
Figure 4-22. Density WINDOW STEP IGNORE .....	162
Figure 4-23. Density WINDOW STEP WRAP .....	162
Figure 4-24. Density INSIDE OF EXTENT .....	163
Figure 4-25. Density INSIDE OF $x1\ y1\ x2\ y2$ .....	163
Figure 4-26. Density INSIDE OF LAYER .....	164
Figure 4-27. Density INSIDE OF LAYER BY EXTENT .....	165
Figure 4-28. Density INSIDE OF LAYER BY POLYGON .....	166
Figure 4-29. Density INSIDE OF LAYER BY RECTANGLE .....	167
Figure 4-30. Density INSIDE OF LAYER CENTERED .....	167
Figure 4-31. GRADIENT Windows .....	169
Figure 4-32. MAGNITUDE Windows .....	170
Figure 4-33. COMBINE Output .....	172
Figure 4-34. Bounding Box and WINDOW Coordinates .....	174
Figure 4-35. TRUNCATE Keyword .....	185
Figure 4-36. WRAP Keyword .....	186
Figure 4-37. Example of a Layer Before and After Density Convolution.....	193
Figure 4-38. Adjustable Resistor .....	203
Figure 4-39. TEXT MODEL LAYER Example .....	212
Figure 4-40. Port-aware Open CAA .....	233

Figure 4-41. DFM Compress COLOR Output .....	241
Figure 4-42. Flat Example 1 .....	244
Figure 4-43. Flat Example 2 .....	244
Figure 4-44. Hierarchical Example .....	245
Figure 4-45. Redundancy on Metal Layers .....	246
Figure 4-46. Critical Area Regions for SHORT .....	258
Figure 4-47. Single-Layer Short Critical Area Model.....	260
Figure 4-48. Isolated Open Via Critical Area Model .....	261
Figure 4-49. Open Via Critical Area Mode for Via Arrays.....	261
Figure 4-50. Single-Layer Open Critical Area Model.....	262
Figure 4-51. OPENWIRE Versus OPENWIRE .....	263
Figure 4-52. LINEENDMAX Versus SIDEMAX.....	281
Figure 4-53. Enclosure Configurations .....	284
Figure 4-54. DFM Fill COLOR Output.....	285
Figure 4-55. Typical Use Model for DFM Fill .....	287
Figure 4-56. How DFM Grow Processes Your Data.....	306
Figure 4-57. DFM Property CORNER .....	324
Figure 4-58. DFM Property SPLIT Behavior .....	329
Figure 4-59. DFM Property SPLIT and SPLIT ALL Projection.....	330
Figure 4-60. DFM Spec Fill PARTITION Example .....	371
Figure 4-61. Relating Fill Extents to Effort.....	379
Figure 4-62. DFM Spec Fill SPACE Options .....	380
Figure 4-63. Corner-to-Corner Spacing Metric Options (DFM Spec Fill) .....	382
Figure 4-64. Fillshape for Example 2 .....	385
Figure 4-65. Fill for Example 3 .....	386
Figure 4-66. Corner-to-Corner Spacing Metric Options .....	402
Figure 4-67. DFM Transform Operation .....	414
Figure 4-68. Possible Via Configurations .....	418
Figure 4-69. Exterior and Interior Cycles .....	428
Figure 4-70. Donut .....	429
Figure 4-71. Shown Acute Angles.....	430
Figure 4-72. Angled Output.....	431
Figure 4-73. Off-Grid Vertices .....	432
Figure 4-74. Skew Edge. ....	433
Figure 4-75. Enclose .....	504
Figure 4-76. Basic Enclosure Rule Checks .....	506
Figure 4-77. Typical Enclosure Check .....	507
Figure 4-78. Poly Extension Check .....	508
Figure 4-79. Measurement Regions .....	511
Figure 4-80. Polygon Containment Criteria Relaxed .....	513
Figure 4-81. Connectivity-Based Checks .....	514
Figure 4-82. Default Edge Orientations Checked by Enclosure .....	515
Figure 4-83. Edge Orientations .....	516
Figure 4-84. Definition of Edge Projection .....	518
Figure 4-85. ANGLED .....	520

## List of Figures

---

Figure 4-86. CORNER TO CORNER .....	521
Figure 4-87. CORNER TO EDGE .....	522
Figure 4-88. ABUT Examples .....	523
Figure 4-89. SINGULAR Examples .....	524
Figure 4-90. INSIDE ALSO and OUTSIDE ALSO .....	526
Figure 4-91. REGION Examples .....	529
Figure 4-92. Expand Cell Text cell_C cell_A .....	559
Figure 4-93. Expand Edge Operation .....	562
Figure 4-94. Expand Edge With CORNER FILL .....	563
Figure 4-95. Extent .....	566
Figure 4-96. Bulk = EXTENT .....	566
Figure 4-97. Extents .....	575
Figure 4-98. Basic External Rule Checks .....	576
Figure 4-99. Typical External Check .....	577
Figure 4-100. Measurement Regions .....	579
Figure 4-101. NOTCH and SPACE .....	581
Figure 4-102. MEASURE ALL .....	582
Figure 4-103. Connectivity-Based Checks .....	583
Figure 4-104. Default Edge Orientations Checked by External .....	584
Figure 4-105. Edge Orientations .....	585
Figure 4-106. ANGLED .....	588
Figure 4-107. CORNER TO CORNER .....	589
Figure 4-108. CORNER TO EDGE .....	590
Figure 4-109. ABUT .....	591
Figure 4-110. External: Measurement and Output at Singularities .....	593
Figure 4-111. INSIDE ALSO Output .....	595
Figure 4-112. REGION Examples .....	599
Figure 4-113. Non-Orientable Polygon .....	605
Figure 4-114. Non-Simple, Non-Orientable Path Expansion .....	607
Figure 4-115. Edge Classification of the Grow Operation .....	633
Figure 4-116. Grow Operation Example .....	633
Figure 4-117. Grow SEQUENTIAL Example .....	635
Figure 4-118. Holes Operation .....	640
Figure 4-119. Holes metal .....	641
Figure 4-120. Holes metal INNER .....	642
Figure 4-121. Holes metal EMPTY .....	642
Figure 4-122. Holes metal INNER EMPTY .....	643
Figure 4-123. Inside .....	651
Figure 4-124. Inside Edge .....	656
Figure 4-125. Interact .....	660
Figure 4-126. Interact BY NET .....	661
Figure 4-127. Basic Internal Rule Checks .....	662
Figure 4-128. Typical Internal Check .....	663
Figure 4-129. Metal Width in Poly Overlap Check .....	664
Figure 4-130. Measurement Regions .....	666

Figure 4-131. MEASURE ALL . . . . .	668
Figure 4-132. Connectivity-Based Checks . . . . .	669
Figure 4-133. Default Edge Orientations Checked by Internal . . . . .	670
Figure 4-134. Edge Orientations . . . . .	671
Figure 4-135. ANGLED . . . . .	674
Figure 4-136. CORNER TO CORNER . . . . .	675
Figure 4-137. CORNER TO EDGE . . . . .	676
Figure 4-138. ABUT . . . . .	678
Figure 4-139. Internal: Measurement and Output at Singularities . . . . .	679
Figure 4-140. REGION Examples . . . . .	682
Figure 4-141. Label Order . . . . .	683
Figure 4-142. Rotated Placements . . . . .	715
Figure 4-143. Transformed Versus Rotated Placements . . . . .	717
Figure 4-144. Acute Angle Path Expansion . . . . .	749
Figure 4-145. Path Expansion for Short End Segments . . . . .	749
Figure 4-146. Layout Windel . . . . .	812
Figure 4-147. Layout Windel with Layout Window Clip YES . . . . .	812
Figure 4-148. Layout Window . . . . .	817
Figure 4-149. Layout Window with Layout Window Clip YES . . . . .	817
Figure 4-150. Layout Window Cell . . . . .	819
Figure 4-151. Layout Windel Cell . . . . .	819
Figure 4-152. Layout Window Layer . . . . .	821
Figure 4-153. Layout Windel Layer . . . . .	822
Figure 4-154. Length . . . . .	823
Figure 4-155. Well-Defined Center Device Pin Location . . . . .	860
Figure 4-156. Interaction Edge Pin Location . . . . .	861
Figure 4-157. Arbitrary Device Pin Location for Irregular Polygons . . . . .	861
Figure 4-158. Undefined Centered Pin Location . . . . .	862
Figure 4-159. Unused Device Filter Option Examples . . . . .	909
Figure 4-160. Redundant Short Elimination . . . . .	934
Figure 4-161. No Shorts in Any Cell . . . . .	937
Figure 4-162. Seed Promotion . . . . .	984
Figure 4-163. LVS Reduction Priority . . . . .	1033
Figure 4-164. SPLIT Option Behavior . . . . .	1053
Figure 4-165. Seed Promotion . . . . .	1058
Figure 4-166. LVS Split Gate Ratio, Example 1 . . . . .	1102
Figure 4-167. LVS Split Gate Ratio, Example 2 . . . . .	1103
Figure 4-168. LVS Split Gate Ratio, Example 3 . . . . .	1104
Figure 4-169. Data With Map Layers . . . . .	1133
Figure 4-170. MDP Mapsize SVRF Example . . . . .	1134
Figure 4-171. Soft Boundary Behavior Example . . . . .	1135
Figure 4-172. Illustration of MDP MASKOPT Arguments . . . . .	1139
Figure 4-173. Merge . . . . .	1148
Figure 4-174. NET AREA RATIO metal1 gate > 20 . . . . .	1153
Figure 4-175. Net Area Ratio Accumulation . . . . .	1171

## List of Figures

---

Figure 4-176. Net Interact . . . . .	1179
Figure 4-177. Boolean NOT . . . . .	1180
Figure 4-178. Not Area . . . . .	1183
Figure 4-179. Not Coincident Edge . . . . .	1184
Figure 4-180. Not Coincident Inside Edge . . . . .	1185
Figure 4-181. Not Coincident Outside Edge . . . . .	1186
Figure 4-182. Not Cut . . . . .	1188
Figure 4-183. Not Donut . . . . .	1189
Figure 4-184. Not Enclose . . . . .	1191
Figure 4-185. Not Inside . . . . .	1194
Figure 4-186. Not Inside Edge . . . . .	1198
Figure 4-187. Not Interact . . . . .	1201
Figure 4-188. Not Length . . . . .	1202
Figure 4-189. Not Outside . . . . .	1204
Figure 4-190. Not Outside Edge . . . . .	1205
Figure 4-191. ASPECT . . . . .	1206
Figure 4-192. Not Rectangle . . . . .	1208
Figure 4-193. Not Touch . . . . .	1210
Figure 4-194. Not Touch Edge . . . . .	1211
Figure 4-195. Not Touch Edge ENDPOINT ALSO . . . . .	1212
Figure 4-196. Not Touch Edge ENDPOINT ONLY . . . . .	1212
Figure 4-197. Not Touch Inside Edge . . . . .	1213
Figure 4-198. Not Touch Inside Edge ENDPOINT ALSO . . . . .	1214
Figure 4-199. Not Touch Inside Edge ENDPOINT ONLY . . . . .	1214
Figure 4-200. Not Touch Outside Edge . . . . .	1215
Figure 4-201. Not Touch Outside Edge ENDPOINT ALSO . . . . .	1216
Figure 4-202. Not Touch Outside Edge ENDPOINT ONLY . . . . .	1216
Figure 4-203. Not With Edge . . . . .	1217
Figure 4-204. Halo Region Using Euclidean Metric . . . . .	1219
Figure 4-205. Halo Region Using SQUARE Metric . . . . .	1220
Figure 4-206. Measurement Region Using CENTERS . . . . .	1220
Figure 4-207. Example of Not With Neighbor . . . . .	1222
Figure 4-208. Output Without CENTERS . . . . .	1227
Figure 4-209. CENTERS With Polygon Input . . . . .	1228
Figure 4-210. Edge Input Without CENTERS . . . . .	1229
Figure 4-211. Edge Input With CENTERS . . . . .	1229
Figure 4-212. INSIDE OF LAYER With Polygon Input . . . . .	1230
Figure 4-213. OFFSET With Edge Input . . . . .	1231
Figure 4-214. Polygon Input With HINT . . . . .	1232
Figure 4-215. Edge Input With HINT . . . . .	1232
Figure 4-216. Opcbias GRID Option . . . . .	1244
Figure 4-217. Via Oversizing . . . . .	1246
Figure 4-218. Line-End Extension . . . . .	1248
Figure 4-219. Serif Dimensions . . . . .	1249
Figure 4-220. Corner Filling . . . . .	1250

Figure 4-221. Types of SPACE Violations . . . . .	1251
Figure 4-222. Applying Correction Rules . . . . .	1253
Figure 4-223. Partial-Cut Negative Scattering Bars . . . . .	1260
Figure 4-224. INTERSECTION Options . . . . .	1260
Figure 4-225. INTERSECTION N Cleanup With and Without OPENT . . . . .	1262
Figure 4-226. Measuring Jog Overlaps . . . . .	1263
Figure 4-227. Styles of Jog Interaction . . . . .	1263
Figure 4-228. CENTERMERGE . . . . .	1264
Figure 4-229. COLINEARIZE . . . . .	1266
Figure 4-230. Using NOCLEANUP . . . . .	1268
Figure 4-231. Typical SPACE Rule . . . . .	1269
Figure 4-232. Typical SPACELAYER Rule . . . . .	1269
Figure 4-233. WIDTH for Positive Scattering Bars . . . . .	1270
Figure 4-234. SBPITCH . . . . .	1271
Figure 4-235. CENTER . . . . .	1272
Figure 4-236. CENTERPITCH . . . . .	1273
Figure 4-237. Rule Types . . . . .	1275
Figure 4-238. Two-Layer Boolean OR . . . . .	1282
Figure 4-239. OR Edge . . . . .	1283
Figure 4-240. Gate Edges . . . . .	1284
Figure 4-241. Outside . . . . .	1286
Figure 4-242. Outside Edge . . . . .	1287
Figure 4-243. Path Length . . . . .	1292
Figure 4-244. Pathchk BREAKNAME . . . . .	1302
Figure 4-245. Perimeter . . . . .	1319
Figure 4-246. Metal1-Metal2 Capacitor Layout . . . . .	1324
Figure 4-247. Two Parallel M1-M2 Capacitors . . . . .	1324
Figure 4-248. Parasitic Interaction Between M1-M2 Capacitors . . . . .	1325
Figure 4-249. Simple Inverter Layout With Ideal Bulk and N-Well Taps . . . . .	1336
Figure 4-250. Cross Section of Ideal PSUB Tap for NMOS . . . . .	1336
Figure 4-251. Non-Ideal PSUB Tap for NMOS . . . . .	1337
Figure 4-252. Model of Extraction Without PEX Bulk Layer . . . . .	1337
Figure 4-253. Model of Extraction With PEX Bulk Layer . . . . .	1338
Figure 4-254. Simple Floating Net Example . . . . .	1360
Figure 4-255. Extraction with Resistance and Coupled Capacitance (-rcc) . . . . .	1361
Figure 4-256. Extraction with Resistance and Capacitance (-rc) . . . . .	1361
Figure 4-257. MOSFET With Gate Resistor, Rg, and Valid Gate Shape . . . . .	1365
Figure 4-258. Single-Sided and Double-Sided Gate Connection . . . . .	1365
Figure 4-259. Double-Sided Gate Connection With Interconnect Loop . . . . .	1366
Figure 4-260. Double-Sided Gate Connection With Delta Model . . . . .	1366
Figure 4-261. Gate Resistance Calculation Example . . . . .	1367
Figure 4-262. Top View of Cell for Boundary . . . . .	1372
Figure 4-263. Top View of Cell for Internal Boundary . . . . .	1373
Figure 4-264. Boundary Locations Using BY_ENCLOSURE . . . . .	1374
Figure 4-265. BOUNDARY Locations Using BY_EXTENT . . . . .	1375

## List of Figures

---

Figure 4-266. Top View of Cell For Cell_array .....	1377
Figure 4-267. Cell array Using REFLECTIVE(X) and PERIODIC(Y) Settings .....	1378
Figure 4-268. Top View of Cell With Internal Boundary .....	1378
Figure 4-269. Top View of Cell With Default Boundary .....	1379
Figure 4-270. Cell_array Boundaries Using BY_ENCLOSURE .....	1380
Figure 4-271. Cell_array Boundaries Using BY_EXTENT and BY_ENCLOSURE .....	1381
Figure 4-272. Cell_array Boundaries Using BY_EXTENT .....	1382
Figure 4-273. Cell_array Boundaries With Floating Metal .....	1383
Figure 4-274. Cell_array Boundaries Using OPEN_BOUNDARY .....	1384
Figure 4-275. Endcap Spacing Distance .....	1385
Figure 4-276. Definition of Capacitance Types .....	1399
Figure 4-277. Definition of Capacitance Types with Vias .....	1399
Figure 4-278. Ignored Capacitance Types for Syntax 1 Example 1 .....	1401
Figure 4-279. Ignored Capacitance Types for Syntax 1 Example 2 .....	1402
Figure 4-280. Ignored Capacitance Types for Syntax 1 Example 3 .....	1402
Figure 4-281. Ignored Capacitance Types with Vias for Syntax 1 Example 4 .....	1403
Figure 4-282. Ignored Capacitance Types for Syntax 2 Example .....	1404
Figure 4-283. Ignored Capacitance Types for Syntax 3 Example .....	1404
Figure 4-284. Ignored Capacitance Types with Vias for Syntax 3 .....	1405
Figure 4-285. Ignored Capacitance Types for Syntax 4 Example 1 .....	1405
Figure 4-286. Ignored Capacitance Types with Vias for Syntax 4 Example .....	1406
Figure 4-287. Ignored Capacitance Types with Vias for Syntax 5 Example .....	1407
Figure 4-288. Ignored Capacitance Types for Syntax 6 Example 1 .....	1407
Figure 4-289. Ignored Capacitance Types with Vias for Syntax 6 Example 2 .....	1408
Figure 4-290. ADITRAIL xRC netlist output example .....	1466
Figure 4-291. Example of 1:N Transistor Mapping .....	1470
Figure 4-292. Example of 1:N Transistor Mapping .....	1506
Figure 4-293. Hierarchical Design and Interconnect .....	1512
Figure 4-294. Net Model Representation of Design .....	1514
Figure 4-295. Coupling Capacitor Example .....	1530
Figure 4-296. Example Circuit Reduced with TICER PORTMERGE .....	1542
Figure 4-297. Flexible Versus Standard Via Reduction .....	1546
Figure 4-298. PEX Report Netsummary Example .....	1558
Figure 4-299. COORD Usage Example .....	1561
Figure 4-300. Inverter Cell with Ideal Bulk and N-Well Taps .....	1566
Figure 4-301. Cross-Section of Ideal PSUB Tap to NMOS Bulk Pin .....	1567
Figure 4-302. Non-Ideal PSUB Tap Found in CELL_1 .....	1567
Figure 4-303. Extracted Network Without Using BULKRESISTANCE .....	1567
Figure 4-304. Extracted Network with BULKRESISTANCE set to 0 .....	1568
Figure 4-305. Extracted Network with Non-Zero BULKRESISTANCE .....	1569
Figure 4-306. Parasitic Extraction with Transistor-Level View of Standard Cell .....	1586
Figure 4-307. Parasitic Extraction with Gray Box Mode .....	1587
Figure 4-308. Parasitic Extraction with Black Box Mode .....	1587
Figure 4-309. Port Layer Polygon .....	1597
Figure 4-310. ASPECT .....	1608

Figure 4-311. Rectangle . . . . .	1610
Figure 4-312. Rectangle Enclosure Check . . . . .	1613
Figure 4-313. Rectangles Parameters . . . . .	1620
Figure 4-314. Rectangles With Default Spacing . . . . .	1621
Figure 4-315. Rectangles With Maintained Spacing . . . . .	1621
Figure 4-316. Example Two-Layer Resistance Connection . . . . .	1626
Figure 4-317. Example Connect By Resistance Connection . . . . .	1627
Figure 4-318. Resistance Sheet Fracturing . . . . .	1632
Figure 4-319. Edge Classification of Shrink Operation . . . . .	1654
Figure 4-320. Shrink . . . . .	1654
Figure 4-321. Oversizing and Undersizing a Polygon . . . . .	1658
Figure 4-322. Worm Rule Check . . . . .	1659
Figure 4-323. Handling of Polygon Spike . . . . .	1661
Figure 4-324. Size BEVEL . . . . .	1662
Figure 4-325. Full-edge Shielding . . . . .	1682
Figure 4-326. Partial-Edge Shielding . . . . .	1683
Figure 4-327. Touch . . . . .	1695
Figure 4-328. Touch Edge . . . . .	1696
Figure 4-329. Touch Edge ENDPOINT ALSO . . . . .	1697
Figure 4-330. Touch Edge ENDPOINT ONLY . . . . .	1697
Figure 4-331. Touch Inside Edge . . . . .	1698
Figure 4-332. Touch Inside Edge ENDPOINT ALSO . . . . .	1699
Figure 4-333. Touch Inside Edge ENDPOINT ONLY . . . . .	1699
Figure 4-334. Touch Outside Edge . . . . .	1700
Figure 4-335. Touch Outside Edge ENDPOINT ALSO . . . . .	1701
Figure 4-336. Touch Outside Edge ENDPOINT ONLY . . . . .	1701
Figure 4-337. Vertex . . . . .	1732
Figure 4-338. Virtual Connect Box Colon . . . . .	1734
Figure 4-339. Virtual Connect Box Colon . . . . .	1734
Figure 4-340. Virtual Connect Box Name . . . . .	1736
Figure 4-341. Virtual Connect Colon YES . . . . .	1738
Figure 4-342. Virtual Connections . . . . .	1748
Figure 4-343. With Edge . . . . .	1755
Figure 4-344. Halo Region Using Euclidean Metric . . . . .	1758
Figure 4-345. Halo Region Using SQUARE Metric . . . . .	1759
Figure 4-346. Measurement Region Using CENTERS . . . . .	1759
Figure 4-347. Angled Region . . . . .	1764
Figure 4-348. One-Layer Boolean XOR . . . . .	1769
Figure 4-349. Two-Layer Boolean XOR . . . . .	1769
Figure 5-1. Computation of Bends . . . . .	1788
Figure 5-2. device::enclosure_measurements Parameters . . . . .	1799
Figure 5-3. device::enclosure_measurements Parameters . . . . .	1801
Figure 5-4. ENCclosure_PARallel() and ENCclosure_PARallel_MULTifinger() . . . . .	1808
Figure 5-5. ENCclosure_PERpendicular() and ENCclosure_PERpendicular_MULTifinger() . . . . .	1808
Figure 5-6. ENCclosure_VECtor() Vector Elements . . . . .	1815

## List of Figures

---

Figure 5-7. Perimeter Relationships .....	1820
Figure 8-1. METAL_WIDTH_CHECK .....	2074
Figure 8-2. gate_dimension_rule .....	2075
Figure 8-3. Width With Overlap .....	2076
Figure 8-4. POLY_DIFF_SEP.....	2079
Figure 8-5. METAL_SPACING .....	2080
Figure 8-6. BC_CORNER_ERROR .....	2081
Figure 8-7. ACUTE ANGLE LENGTH CHECK.....	2083
Figure 8-8. POLY_DIFF_SPACING (part 1).....	2084
Figure 8-9. POLY_DIFF_SPACING (part 2).....	2085
Figure 8-10. Rule7.4 .....	2086
Figure 8-11. poly_to_diff1.....	2087
Figure 8-12. poly_to_diff2.....	2088
Figure 8-13. NARROW_NOTCH.....	2089
Figure 8-14. Rule_53a .....	2089
Figure 8-15. Rule_53b .....	2090
Figure 8-16. PEM.....	2090
Figure 8-17. Rule_987.....	2091
Figure 8-18. POLY_EXTENSION_ERROR.....	2092
Figure 8-19. ABC Check (part 1) .....	2094
Figure 8-20. ABC Check (part 2) .....	2095
Figure 8-21. ABC Check (part 3) .....	2096
Figure 8-22. NON_OVLP_CNT .....	2097
Figure 8-23. I_DIR_SPACING .....	2098
Figure 8-24. WTDC Polygons .....	2099
Figure 8-25. WTDC C_IN .....	2100
Figure 8-26. WTDC C_NOT_IN .....	2100
Figure 8-27. WTDC First Output .....	2101
Figure 8-28. WTDC Second Output .....	2101
Figure 8-29. WTDC C1 and Y.....	2102
Figure 8-30. WTDC Z .....	2102
Figure 8-31. WTDC Third Output .....	2103
Figure 8-32. metal_over_contact .....	2104
Figure 8-33. merge_poly .....	2105
Figure 8-34. Layer Derivations .....	2106
Figure 8-35. Source/Drain Regions .....	2106
Figure 8-36. Find Wide Metal .....	2107
Figure 8-37. Using Magnify to Enlarge a Layer .....	2108
Figure 8-38. Using Magnify to Shrink a Layer .....	2108
Figure 8-39. bad_gates.....	2109
Figure 8-40. contacts_in_holes .....	2110

# List of Tables

---

Table 1-1. Products Discussed in this Manual .....	35
Table 1-2. Syntax Conventions .....	36
Table 2-1. SVRF Symbols .....	41
Table 2-2. Constraints .....	45
Table 2-3. Numeric Operators in Order of Precedence .....	46
Table 2-4. Keyword - Name Conflicts .....	49
Table 2-5. Inlined Keyword/Command Syntax Rules .....	50
Table 2-6. Rule File Elements .....	52
Table 3-1. Layer Operations: Dimensional Check Operations .....	71
Table 3-2. Layer Operations: Auxiliary Operations .....	72
Table 3-3. Connectivity Extraction Operations .....	80
Table 3-4. Specification Statements: Virtual Connect .....	81
Table 3-5. Fracturing Statements .....	82
Table 3-6. Post-Design Correction Statements .....	83
Table 3-7. Specification Statements: DFM .....	84
Table 3-8. Specification Statements: nmDRC .....	85
Table 3-9. Specification Statements: ERC .....	88
Table 3-10. Specification Statements: Layout Database .....	89
Table 3-11. Specification Statements: LVS .....	94
Table 3-12. Specification Statements: PERC .....	104
Table 3-13. Specification Statements: PEX .....	104
Table 3-14. Specification Statements: PEX Inductance .....	110
Table 3-15. Specification Statements: Unit .....	112
Table 3-16. Specification Statements: Other .....	112
Table 4-1. Rule Check Properties .....	149
Table 4-2. Density Math Functions .....	157
Table 4-3. ! and ~ Operator Combinations .....	158
Table 4-4. GRADIENT Values .....	168
Table 4-5. Built-in Devices with Default Properties .....	196
Table 4-6. Built-in Devices Without Default Properties .....	198
Table 4-7. DFM Copy Merging Behavior for Unmerged Input .....	250
Table 4-8. Critical Area Generation Methods .....	257
Table 4-9. Using the Correct Measurement Rule Syntax .....	313
Table 4-10. M1 Width and Spacing (microns) .....	316
Table 4-11. Layer Types Supported by DFM RDB .....	336
Table 4-12. Summary of NODAL Options .....	337
Table 4-13. Rule Check Names for RDBs Generated by DFM RDB .....	340
Table 4-14. Table-offsets Structure .....	483
Table 4-15. Exceptions .....	730
Table 4-16. Layout Preserve Case Keywords .....	771

## List of Tables

---

Table 4-17. Regular Expression Examples . . . . .	792
Table 4-18. Layout Rename Text Editing Examples . . . . .	794
Table 4-19. Layout Input Specification Statement . . . . .	797
Table 4-20. MGC_CALIBRE_DB_READ_OPTIONS Arguments . . . . .	798
Table 4-21. Device Types . . . . .	874
Table 4-22. Built-In Device Pin Names for Mapping . . . . .	875
Table 4-23. EDDM Property Processing . . . . .	885
Table 4-24. BY CELL and BY LAYER Output Options . . . . .	933
Table 4-25. High-Shorted LVS Box Port Resolution Examples . . . . .	962
Table 4-26. Device Pushdown Behaviors . . . . .	975
Table 4-27. Keywords for LVS Report Option . . . . .	1037
Table 4-28. Scaled X Device Default Behavior History . . . . .	1094
Table 4-29. Files Generated by Mask SVDB Directory Options . . . . .	1124
Table 4-30. Net Area Ratio Math Functions . . . . .	1154
Table 4-31. Database Output for Net Area Ratio . . . . .	1164
Table 4-32. Net Area Ratio Accumulate Math Functions . . . . .	1173
Table 4-33. Two-dimensional Rules Table . . . . .	1243
Table 4-34. Translating Rules Tables . . . . .	1243
Table 4-35. Height Conditions and Edges . . . . .	1247
Table 4-36. Summary of OPCsbar Parameters and Defaults . . . . .	1276
Table 4-37. XFORM Keyword Options for PERC Load . . . . .	1306
Table 4-38. Order of Rule Check Execution in PERC . . . . .	1307
Table 4-39. Keywords for PERC Report Option . . . . .	1316
Table 4-40. “Worst Case” Parasitic Values for LPE . . . . .	1332
Table 4-41. Device Name, Net Name, and Parameter Conventions in Netlist . . . . .	1464
Table 4-42. PEX Netlist Keyword Combinations and Results . . . . .	1465
Table 4-43. PEX Netlist: Parameter Calculations for Parallel Transistors . . . . .	1467
Table 4-44. PEX Netlist: Parameter Calculations for Series Transistors . . . . .	1468
Table 4-45. PEX Netlist Distributed Keyword Combinations and Results . . . . .	1484
Table 4-46. PEX Netlist SchematicOnly: Parameter Calculations for Parallel Transistors .	1505
Table 4-47. PEX Netlist SchematicOnly: Parameter Calculations for Series Transistors .	1505
Table 4-48. Nets Reported for Secondary Keywords . . . . .	1558
Table 4-49. Specifying Sheet Resistance and BULKRESISTANCE . . . . .	1565
Table 5-1. Built-In Language General Characteristics . . . . .	1771
Table 5-2. Numeric Functions for Built-in Languages . . . . .	1775
Table 5-3. Property Computation Built-In Language Specifics . . . . .	1779
Table 5-4. Device Property Computation Built-In Functions . . . . .	1782
Table 5-5. Perimeter Functions . . . . .	1820
Table 5-6. Return Values of TEXT_NUMERIC Function . . . . .	1823
Table 5-7. Return Values of TEXT_STRING() Function . . . . .	1825
Table 5-8. Differences in Enclosure Calculations . . . . .	1834
Table 5-9. Device Annotation Built-In Language Specifics . . . . .	1843
Table 5-10. Device Annotation Built-In Functions . . . . .	1847
Table 5-11. Effective Property Language Specifics . . . . .	1855
Table 5-12. LVS Property Initialize Language Specifics . . . . .	1868

Table 5-13. Trace Property Language Specifics . . . . .	1875
Table 5-14. Summary of Trace Property Language Functions . . . . .	1879
Table 5-15. STRING_COMPARE Return Values . . . . .	1882
Table 6-1. Tcl/TVF Versus SVRF Syntax . . . . .	1890
Table 6-2. Specification Statement Families Mapped to TVF Commands1 . . . . .	1894
Table 6-3. Comment Characters for Compile-time TVF . . . . .	1896
Table 6-4. Preprocessor-Only Commands . . . . .	1904
Table 6-5. Runtime TVF Layer Functions . . . . .	1911
Table 6-6. Layer Operations in Runtime TVF . . . . .	1913
Table 6-7. Accessible System Variables . . . . .	1915
Table 6-8. Compile-Time Commands for Namespace Importation . . . . .	1921
Table 6-9. Runtime Commands for Namespace Importation . . . . .	1921
Table 6-10. TVF Commands . . . . .	1924
Table 6-11. Global Variables . . . . .	1932
Table 7-1. Continuous nmDRC Errors . . . . .	1938
Table 7-2. LVS Cell List Statement Errors . . . . .	1938
Table 7-3. Constraint Errors . . . . .	1938
Table 7-4. DRC Check Map Statement Errors . . . . .	1940
Table 7-5. Pre-Processor Directive Errors . . . . .	1943
Table 7-6. Density Operation Errors . . . . .	1943
Table 7-7. Device Definition Errors . . . . .	1945
Table 7-8. DFM Operation Errors . . . . .	1948
Table 7-9. Device Property Specification Errors . . . . .	1959
Table 7-10. LVS Device Type Statement Errors . . . . .	1963
Table 7-11. Device Layer Operation Errors . . . . .	1963
Table 7-12. Environment Variable Errors . . . . .	1964
Table 7-13. ERC Errors . . . . .	1964
Table 7-14. Connectivity Extraction Errors . . . . .	1964
Table 7-15. LVS Filter Statement Errors . . . . .	1966
Table 7-16. Rule Check Group Errors . . . . .	1967
Table 7-17. Included Rule File Errors . . . . .	1967
Table 7-18. Operation Input Errors . . . . .	1967
Table 7-19. Secondary Keyword Errors . . . . .	1971
Table 7-20. LVS Annotate Devices Errors . . . . .	1973
Table 7-21. Layer Definition Errors . . . . .	1973
Table 7-22. Layout Cell List Statement Errors . . . . .	1974
Table 7-23. Layout Cell Match Rule Statement Errors . . . . .	1974
Table 7-24. Layout MOS Swappable Properties Statement Errors . . . . .	1974
Table 7-25. Location Errors . . . . .	1975
Table 7-26. LVS Push Devices Statement Errors . . . . .	1975
Table 7-27. LVS Property Initialize Statement Errors . . . . .	1975
Table 7-28. PERC Load Statement Errors . . . . .	1977
Table 7-29. LVS Property Map Statement Errors . . . . .	1977
Table 7-30. PERC Property Statement Errors . . . . .	1979
Table 7-31. LVS Reduce Statement Errors . . . . .	1979

## List of Tables

---

Table 7-32. LVS Short Equivalent Nodes Errors . . . . .	1981
Table 7-33. Litho Operation Errors . . . . .	1981
Table 7-34. Macro Errors . . . . .	1983
Table 7-35. LVS Map Device Statement Errors . . . . .	1983
Table 7-36. Net Area Ratio Errors . . . . .	1985
Table 7-37. Numeric Errors . . . . .	1987
Table 7-38. Layer Statement Errors . . . . .	1989
Table 7-39. OPC Operation Errors . . . . .	1989
Table 7-40. General File I/O Errors . . . . .	1992
Table 7-41. Rule Check Statement Errors . . . . .	1992
Table 7-42. Parasitic Extraction Errors . . . . .	1992
Table 7-43. PEX Statement Errors . . . . .	1996
Table 7-44. PEX Property Computation Errors . . . . .	1997
Table 7-45. RDB Conflict Errors . . . . .	1999
Table 7-46. Layer and Variable Resolution Errors . . . . .	1999
Table 7-47. LVS Recognize Gates Tolerance Statement Errors . . . . .	2001
Table 7-48. LVS Split Gate Ratio Statement Errors . . . . .	2001
Table 7-49. General Specification Statement Errors . . . . .	2003
Table 7-50. LVS Spice Rename Parameter Statement Errors . . . . .	2005
Table 7-51. Syntax Errors . . . . .	2005
Table 7-52. Trace Property Statement Errors . . . . .	2006
Table 7-53. Layer Type Errors . . . . .	2008
Table 7-54. SVRF Error Statement . . . . .	2008
Table 7-55. Variable Statement Errors . . . . .	2008
Table 7-56. SVRF Version Statement . . . . .	2009
Table 7-57. Calibre nmDRC-H Warnings . . . . .	2011
Table 7-58. Calibre File I/O Errors . . . . .	2013
Table 7-59. Calibre Precision Exceptions . . . . .	2013
Table 7-60. Calibre Cell and Placement Exceptions . . . . .	2015
Table 7-61. Calibre Layout Input Errors . . . . .	2017
Table 7-62. Calibre Layout Input Exceptions and Warnings . . . . .	2027
Table 7-63. Calibre GDSII and OASIS Output Warnings . . . . .	2034
Table 7-64. Calibre Output Warnings . . . . .	2036
Table 7-65. Cell Name/Location Warnings . . . . .	2036
Table 7-66. Connectivity Extraction and Stamp Warnings . . . . .	2038
Table 7-67. DFM Warnings . . . . .	2040
Table 7-68. ICrules Input Warnings . . . . .	2042
Table 7-69. ICrules Notes . . . . .	2042
Table 7-70. LVS Connectivity Extraction Messages . . . . .	2044
Table 7-71. LVS Comparison Messages . . . . .	2048
Table 7-72. LVS Stamp Error Messages . . . . .	2049
Table 7-73. PERC TVF Errors . . . . .	2050
Table 7-74. Maximum Results and Polygon Segmentation Warnings . . . . .	2056
Table 7-75. Miscellaneous Warnings . . . . .	2056
Table 7-76. Output File Open Warnings . . . . .	2058

Table 7-77. Polygon Flagging Warnings .....	2060
Table 7-78. Rule File Errors .....	2062
Table 7-79. Unsupported Functionality Errors and Warnings .....	2063

# Chapter 1 Preface

---

This manual describes SVRF syntax, operations and specification statements used with Calibre® and ICverify™ products, and contains key concepts about rule files. The manual describes the SVRF syntax elements including the Calibre 2011.3 release. Some new features may not be described in this manual. For the latest functionality, refer to the [Calibre Release Notes](#). The table below lists the products that are referred to in this manual.

**Table 1-1. Products Discussed in this Manual**

Calibre® ADP	Calibre® PERC
Calibre® nmDRC	Calibre® PRINTimage™
Calibre® nmDRC-H™	Calibre® PSMgate™
Calibre® FRACTUREj™	Calibre® RVE™
Calibre® FRACTUREm™	Calibre® TDopc™
Calibre® FRACTUREt™	Calibre® WORKbench™
Calibre® Interactive™	Calibre® xRC™
Calibre® nmLVS	Calibre® xL
Calibre® nmLVS-H™	Calibre® YieldAnalyzer
Calibre® MDPmerge™	Calibre® YieldEnhancer
Calibre® MDPstat™	Pyxis®
Calibre® MDPverify™	Pyxis Layout™
Calibre® OPCsbar™	ICrules™
Calibre® OPCpro™	ICtrace™
Calibre® ORC	ICverify™

Refer to the [Calibre Administrator's Guide](#) for complete information.

## Audience

This manual is written for anyone writing rule files for use with Calibre and ICverify tools to perform layout verification. This manual assumes that you are already familiar with the general layout verification requirements for your designs. If you are using Pyxis Layout, this manual assumes you are already familiar with the tool.

# Documentation Conventions

The brief summary of Mentor Graphics documentation conventions provided here should facilitate your reading of all Mentor Graphics manuals. These conventions apply to functions, commands, key names, pathnames, menu paths, transcripts, and notational information, as described in the sections that follow.

## Rule File Statements

The notational elements used in this manual are shown in [Table 1-2](#). You should enter literal text, that which is not in italics, exactly as shown.

**Table 1-2. Syntax Conventions**

Convention	Description
<b>Bold</b>	A bold font indicates a required item.
<i>Italic</i>	An italic font indicates a user-supplied argument.
Monospace	A monospace font indicates a shell command, line of code, or URL. A bold monospace font identifies text to be entered by the user.
Underline	An underlined item indicates either the default argument or the default value of an argument.
UPPercase	Uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword. Also see “ <a href="#">Keyword Abbreviations</a> ” on page 48.
[ ]	Square brackets enclose optional arguments. Do not include the brackets when entering the command.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command.
‘ ’	Single quotes enclose metacharacters that are intended to be entered literally.
	The vertical bar indicates an either/or choice between items. Do not include the bar when entering the command.
...	An ellipsis follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.
<b>Example:</b>	
<pre>DEvice {element_name [('model_name')]}     device_layer {pin_layer ['pin_name'] ...}     [&lt;'auxiliary_layer'&gt; ...] ['swap_list' ...]     [BY NET   BY SHAPE]</pre>	

## Functions

Mentor Graphics application functions are structured like this example:

```
$function_name(arg_1, arg_2, arg_3, arg_4)
```

The following documentation conventions apply to functions, as demonstrated by the preceding example:

- The name of the function always begins with a dollar sign (\$) or, sometimes for built-in functions, two dollar signs. In this example, “\$function\_name” is the name of the function.
- Standard font indicates that an argument is required. In this example, arg\_1 and arg\_2 are required arguments.
- Italic font indicates that an argument is optional. In this example, arg\_3 and arg\_4 are optional arguments.

For the syntax requirements that relate to functions, refer to the “Function Call Syntax and Semantics” section in the *AMPLE User's Manual*.

## Pathnames

Most Mentor Graphics pathnames use shell environment variables, such as \$MGC\_GENLIB and \$PROJECT\_XYZ. The pathnames generally point to shared network resources. The dollar sign indicates dereferencing of the environment variable, and environment variable itself is shown in capital letters, in keeping with the industry-standard practice. This is a typical example:

```
$MGC_HOME/pkgs/dme/userware/default
```

## Notational Information

Mentor Graphics documentation emphasizes notational information by setting it apart from the standard text:

- Cautions, which flag hazardous situations that might result in data loss, program failure, application error, or equipment damage.
- Notes, which flag nonhazardous information that nonetheless requires special attention.



# Chapter 2

## Key Concepts

---

The Mentor Graphics IC verification products—Calibre and ICverify—provide a complete layout verification solution that encompasses:

- Design rule checks (DRC) including Design for Manufacturability (DFM)
- Electrical rule checks (ERC)
- Short isolation
- Soft connection checks
- Layout versus schematic comparison (LVS)
- Parasitic extraction (PEX) and analysis
- Resolution Enhancement Technologies (RET)
- Mask Data Preparation (MDP/Fracture)

Calibre provides functionality for nmDRC, DFM, nmLVS, ERC, PERC, RET, MDP/Fracture, and PEX. ERC, short isolation, and soft connection checks are performed as a part of nmLVS connectivity extraction when these options are enabled. DFM, RET, MDP, and Fracture applications are built on the nmDRC engine.

The ICrules (DRC) and ICtrace (LVS) elements of ICverify provide flat Calibre nmDRC and nmLVS functionality within Pyxis Layout. Calibre xRC CB provides flat parasitic extraction functionality.

When there is no need to distinguish between applications in Calibre, the term Calibre is used. When there is no need to distinguish between Calibre nmDRC, nmDRC-H, and ICrules, the term nmDRC is used. Similarly for Calibre nmLVS, nmLVS-H, and ICtrace.

Major sections of this chapter include the following:

<b>Rule File Overview</b> .....	<b>40</b>
<b>General Syntactic Conventions</b> .....	<b>41</b>
SVRF Symbols and Command Parsing .....	41
Comments .....	44
Constraints .....	45
Numeric Expressions .....	46
Negative Numbers in Coordinate Lists.....	46
String Constants .....	46
Variables .....	47

Environment Variables in Pathname Parameters . . . . .	47
Keyword Abbreviations . . . . .	48
Keyword - Name Conflicts . . . . .	49
Rules for Inlined Parameter Blocks . . . . .	50
<b>Basic Rule File Structure . . . . .</b>	<b>52</b>
Rule File Elements . . . . .	52
Pre-Processor Directives . . . . .	56
Macros . . . . .	64

## Rule File Overview

The Calibre and ICVerify products use a *Standard Verification Rule Format* (SVRF) file (or *rule file* for short). The rule file governs the following activities: original layer definition, derived layer generation, design rule checking, connectivity extraction, electrical rule checking, device recognition, circuit comparison, and parasitic extraction.

All functionality of the Calibre and ICVerify products depend on declarations specified in the rule file. All elements in a rule file can be placed in one of two categories: *operations* and *specification statements*. The basic distinction between categories is that operations work on the layout data and specification statements govern the environment in which the operations function.

Operations and specification statements are further organized into functional groups as shown in the following list. As indicated, each group is explained in its own section of this manual.

- Operations
  - “[Layer Operations](#)” on page 69
  - “[Connectivity Extraction Statements](#)” on page 80
  - “[Device Recognition Statement](#)” on page 82
  - “[Fracturing Statements](#)” on page 82
  - “[Post-Design Correction Statements](#)” on page 83
- “[Specification Statements](#)” on page 84

Each section provides a description of the group, and a list of rule file operations/statements (hereafter called rule file elements, or simply elements) that fit into that group.

You can obtain comprehensive information on each element by referring to the “[Reference Dictionary](#).”

Calibre also accepts a compile-time Tcl Verification Format (TVF) script as a rule file input. See “[Compile-Time TVF](#)” on page 1891 for information about TVF scripts.

## Compilation Overview

Prior to its use by a verification application, the rule file must be compiled. This is automatic in Calibre applications and is performed by the LOAd RUles command in Pyxis Layout.

Compilation involves checking that all syntax is correct, checking that all layers are of the correct type for a particular operation, and so forth. This process results in a compiled image of the rule file that is suitable for execution of its constituent operations by the applications.

Compilation can resolve all dependencies between statements and operations without imposing an artificial ordering on these entities. For this reason (with two exceptions: variable definitions and incremental connectivity) there are no restrictions in the rule file concerning the order of any operation, layer definition, rule check statement, and so forth. This is an important attribute of SVRF rule files.

Sections of the rule file can be conditionally compiled. See “[Conditional Directives for Process Control](#)” on page 56.

## General Syntactic Conventions

At the lowest level, all entities in the rule file (except pre-processor directives, macros, and **Include** statements, processed as the first phase of compilation) are either *keywords*, *numbers*, *string constants*, or *names*. The following sections discuss the details for command parsing, numeric expressions and constraints, and variable usage.

## SVRF Symbols and Command Parsing

During compilation, all keywords that are *symbols* from the following set are recognized immediately regardless of the absence of surrounding whitespace.

**Table 2-1. SVRF Symbols**

Character	Meanings
//	Rule file comment.
@	Rule check comment.
/* */	Block comment delimiters.
=	Assignment character. Used for layer derivations and in certain built-in languages for assignment of variable values.
{ }	Rule check body and Macro delimiter. Delimiter for certain lists of parameters.
“ ”	String constant delimiter.
‘ ’	String constant delimiter.

**Table 2-1. SVRF Symbols (cont.)**

Character	Meanings
( )	Grouping operator for math expressions. Dimensional check operator for negative edge output. Argument list delimiter for built-in language functions. Argument delimiter for device model names and certain other device elements.
[ ]	Built-in language script delimiter. Filename delimiter. Dimensional check operator for positive edge output.
<	Less than. Used in constraints and conditional statements. Left-hand delimiter for layer names in certain cases.
>	Greater than. Used in constraints and conditional statements. Right-hand delimiter for layer names in certain cases.
==	Logical equivalence. Used in constraints and certain conditional statements.
<=	Less than or equal to. Used in constraints and conditional statements.
>=	Greater than or equal to. Used in constraints and conditional statements.
!=	Not equal to. Used in constraints and conditional statements.
- + * /	Mathematical operators with customary meanings. The * is a wildcard matching 0 or more instances for cell names. Strings that use wildcards should be enclosed in quotation marks.
^	Exponentiation operator in certain mathematical expressions. Escape metacharacter for use in rule check (@) comments to dereference the value of a variable name used in the comment.
!	Logical negation. Used in certain mathematical expressions.
~	Bitwise complement. Returns 0 for positive arguments and 1 for non-positive arguments. Used in certain mathematical expressions.
%	Modulus operator. Used in certain mathematical expressions.
&&	Logical AND operator. Used in certain LVS statements.
	Logical OR operator. Used in certain LVS statements.
::	Scoping operator in TVF and other Tcl based extensions to SVRF. Also used in some built-in languages as a vector operator.
,	Argument separator. Used in built-in language functions.

**Table 2-1. SVRF Symbols (cont.)**

Character	Meanings
?	Wildcard matching 0 or more characters for net names. Strings that use wildcards should be enclosed in quotation marks.
\$	Environment variable dereferencing operator in pathnames.
#	Pre-processor conditional statement definition operator.

Many of these symbols are discussed in greater detail within the appropriate contexts throughout the manual.

Comments are processed after symbols, along with string constants (described later).

The next items processed are remaining SVRF keywords such as “and” or “parallel.” SVRF keywords are always *case-insensitive*. Cell names and filenames are case-sensitive. Net names may or may not be case-sensitive, depending on the settings in the rule file. In general, all other elements in an SVRF rule file are not case-sensitive but some can be made so by certain SVRF statements.

Every operation and specification statement has one or more constituent keywords that describe the operation or statement. Some keywords consist of more than one word; in this case, the words must be separated by whitespace (excluding new lines). One keyword in each operation or statement is designated as *primary* since it identifies the name of the operation. Examples of primary keywords are Group, Area, External, and Not Outside. Examples of keywords that are *secondary* are PARALLEL and ACUTE. Again, keywords are not case-sensitive, but secondary keywords are shown in uppercase in this manual to distinguish them from the primary keywords.

*Numbers* are recognized after SVRF keywords.

*Names* are recognized last. In general, user-specified *names* may be composed of letters, numbers, periods (.), and underscores (\_). They should not contain special symbols contained in **Table 2-1** nor should they contain keyword names specified in **Table 2-4**. However, names that appear as string constants can override these limitations. See “[String Constants](#)” for more details.

The symbols *layer*, *layer1*, *layer2*, … *layerN*, that appear in the syntax denote either names of original or derived layers, or numbers of original layers. The layer type (original, derived edge, or derived polygon) is normally context-sensitive; usage restrictions are discussed where appropriate.

The syntax elements of an operation (but not in a specification statement) can generally appear in any order. This freedom is intended to accommodate various user preferences. However, you cannot break up or rearrange individual syntax elements that consist of multiple words. For example, the operation “Not Inside” cannot have constituents inserted between the Not and the Inside, nor can it be rearranged as Inside Not.

The following six [Area](#) operations are equivalent:

```
area < 4 contact
contact area < 4
area contact < 4
< 4 area contact
< 4 contact area
contact < 4 area
```

as are the following three [OR](#) operations:

```
metal or poly
metal poly or
or metal poly
```

However, there are cases where you should be careful with the ordering of the constituents in an operation. One such instance is a non-commutative operation involving two input layers; *layer1* is always identified with the first input layer name encountered in the operation. For example, the following two [Inside](#) operations are generally not equivalent:

```
contact inside metal
metal inside contact
```

Other cases where ordering can be important are documented as appropriate with the descriptions of the operations to follow.

Finally, the statements and operations that appear in a rule file are not generally sensitive to whitespace; that is, they can begin anywhere on a line and can span lines (continuation characters are not used). In addition, statements and operations do not each need to begin on a new line. If the boundary between two consecutive operations seems unclear, then it is set at the primary keyword of the second operation when both primary keywords are on the same line; otherwise, it is set at the beginning of the line containing the primary keyword of the second operation.

## Comments

Comments are permitted in the rule file to annotate the source code. The // characters start a comment that terminates at the end of the line on which they occur. Multi-line C-style comments /\* ... \*/ are also permitted. See “[Pre-Processor Directives](#)” on page 56 for information on compilation order of comments.

There is also a rule check comment, which is preceded by the @ symbol. These start a comment that terminates at a new line and appear within rule checks. These comments are used as debugging aids and appear as comments in Pyxis Layout and Calibre RVE for DRC when you view rule check results in these interfaces.

Calibre RVE for DRC also recognizes certain rule check comments as configuration commands for highlighting and display preferences; see “[DRC Rule Check Comments for Calibre RVE](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## Constraints

Certain layer operations are measurement-oriented and therefore carry constraints. *Constraints* are ranges of non-negative real numbers; input data that falls within the specified constraint of an operation is generally output data. The syntax for constraints uses one of the six keywords (operators) shown in Table 2-2.

Notice that the syntax for the last four forms is simply a *combination* of that for the first four forms, in most cases,  $a \geq 0$  and  $a < b$ . The constraint “ $< 0$ ” is not allowed because strictly negative constraint values are not possible in SVRF operations. The constraint “ $\leq 0$ ” is permitted.

As an example, the constraint “ $< 4$ ” denotes all non-negative numbers less than 4, and the constraint “ $\geq 5 < 7$ ” denotes all numbers greater than or equal to 5 and less than 7.

Not all operations accommodate all types of constraints or all values of the numbers  $a$  and  $b$ . For example,  $a = 0$  in the following constraint is not valid for use with the [With Width](#) operation, because a polygon with zero width is not possible for this operation.

```
WITH WIDTH layer1 >= a < b // 0 not valid for a
```

Restrictions are discussed, when applicable, with the description of the operation.

Constraints that have two numeric values, such as the final four forms in [Table 2-2](#), are called *interval constraints*.

**Table 2-2. Constraints<sup>1</sup>**

Operators	Constraint Notation	Alternate Constraint Notation	Mathematical Notation
<	< a		$x < a$
>	> a		$x > a$
$\leq$	$\leq a$		$x \leq a$
$\geq$	$\geq a$		$x \geq a$
$\equiv$	$\equiv a$		$x = a$
$\neq$	$\neq a$		$x \neq a$
$>$ and $<$	$> a < b$	$< b > a$	$a < x < b$
$\geq$ and $<$	$\geq a < b$	$< b \geq a$	$a \leq x < b$
$>$ and $\leq$	$> a \leq b$	$\leq b > a$	$a < x \leq b$
$\geq$ and $\leq$	$\geq a \leq b$	$\leq b \geq a$	$a \leq x \leq b$

1.  $a$  and  $b$  are non-negative numbers, and “ $< 0$ ” is not allowed.

## Numeric Expressions

A numeric expression can specify any numeric parameter in any layer operation. Numeric expressions can also be used to define variables in a [Variable](#) specification statement. Numeric expressions may be used as numeric parameters in many specification statements and in functions used by various built-in languages used within SVRF statements.

A numeric expression is a combination of numeric constants, numeric variables, and the operators shown in [Table 2-3](#) in order of precedence from highest to lowest:

**Table 2-3. Numeric Operators in Order of Precedence**

Operator	Operator Type	Definition
( )	grouping	evaluates the numeric value of the enclosed expression
+ -	unary	positive, negative
* / ^ %	binary	multiplication, division, power, modulus
+ -	binary	addition, subtraction

For example, ‘3 \* ( GRID + 6 )’ is a numeric expression here:

```
rule { EXT metal < 3 * ( GRID + 6 ) } // GRID is a numeric variable
```

Note that these operators are supported as a part of the SVRF language, but the entire set of operators may not be supported in all tool-specific languages referenced by SVRF statements, such as the built-in languages for LVS applications.

## Negative Numbers in Coordinate Lists

Some SVRF elements have lists of x y coordinate pairs as parameters. In certain cases, negative numbers must be enclosed in parentheses. For example:

```
e = DENSITY metal < 0.25 ... INSIDE OF (-200) (-100) 400 200
```

If you do not specify parentheses in these cases, then the minus sign is interpreted as subtraction rather than unary minus. This can have unintended results. These cases are discussed under the parameters to which this situation applies.

## String Constants

A *string constant* is a sequence of one or more characters enclosed in quotation marks (“ ”) or single quotes (‘ ’). String constants cannot span lines, and, if the trailing quote is omitted, they extend to the end of the line in the rule file on which they appear. In addition, standard C escape sequences can be embedded. However, you should be careful when embedding whitespace in string constants, especially when naming rule check statements, as it can be difficult (for

example, in the Pyxis Layout environment) to specify these on the command line. Empty string constants are ignored.

A string constant has the important property that it can be used wherever a name is required and, in that context, is treated as a name regardless of the characters that comprise it. This allows names to coincide with SVRF keywords, to contain symbols of the language, or to coincide with numbers.

As an example of the use of string constants:

```
INCLUDE "/user/illamp/block/code/rules"
"inside layer" = x NOT INSIDE y
"523.4" {
EXTERNAL < 4 'inside layer' }
```

## Variables

You can substitute variables in layer operations and many specification statements, but there are restrictions on what can be declared as a variable in SVRF. For more information on declaring variables, refer to the [Variable](#) specification statement.

Environment variables can be referenced using the Variable specification statement. Default values for environment variables can be defined using #PRAGMA ENV directives. See [“Conditional Directive for Environment Variables”](#) on page 62.

---

**Note**



TVF allows greater freedom in the declaration of variables in a rule file. TVF is discussed in detail in the chapter [“Tcl Verification Format”](#) on page 1889.

---

In ICVerify, variables can also be resolved in the Pyxis Layout process file. If a variable is defined both in the process file and in the rule file with a Variable specification statement, then the definitions must match. That is, in ICVerify, Variable specification statements cannot redefine variable names defined in the Pyxis Layout process file. A Pyxis Layout process variable can also be used to define a variable in a Variable specification statement.

When using ICVerify, be aware that Pyxis Layout variables looked up in the process file during compilation of the rule file are resolved to their values *at compilation time*. Therefore, if these values become re-associated while the rule file is loaded, then the loaded rule file does not represent the true state of the Pyxis Layout system and you should reload it.

## Environment Variables in Pathname Parameters

When specifying a pathname parameter in a rule file statement, you can use an environment variable as a component of the filename (note, this does not apply to ICVerify). The environment variable is signified by preceding it with a dollar sign (\$). The tool evaluates the

environment variable at compilation time and substitutes it into the filename string. For example, given the following environment variable definitions:

```
xyz -> /user
pdq -> data
rst -> ram.gds
```

the following specification statement:

```
LAYOUT PATH "$xyz/illamp/$pdq/misc.gds/$rst"
```

is evaluated as:

```
LAYOUT PATH "/user/illamp/data/misc.gds/ram.gds"
```

To make the notion of “filename component” precise, a filename parameter is defined by the following Backus Naur Form (BNF):

```
filename -> [ "/" ] path_str
path_str -> [ "$" ] string [ "/" path_str ]
string -> (any sequence of characters except "/")
```

where \$ designates that the string immediately following is interpreted as an environment variable. Note that a \$variable\_name construct does not generally resolve in a rule file, except in specification statement and layer operation output pathnames, and conditional (IFDEF) statements.

A compilation error (ENV2) occurs when an environment variable required by filename evaluation is undefined, or defined with only a null value.

Pathnames with environment variables may be used in specifying Calibre output files. A pathname declared as an environment variable does not need to be declared in a Variable statement in the rule file.

If a non-existent directory is specified in an output pathname of an SVRF element (such as the RDB keyword for Density or Net Area Ratio operations), then the non-existent directory is created by the Calibre application, assuming the appropriate permissions are used in the environment.

## Keyword Abbreviations

You can abbreviate some of the words in a keyword set. Abbreviations are warranted when the word is long and cumbersome or its expected frequency of use is high. Abbreviations take the form of some prefix of the word to be abbreviated. They are also selected so as not to conflict with common layer names. Abbreviations are generally shown when the keyword is first introduced and are indicated by a subset of the keyword’s letters, which are uppercase.

For example, DEvice indicates the [Device](#) keyword can be abbreviated as Dev.

## Keyword - Name Conflicts

User names (for example, those for layers, rule checks, and cells) cannot generally conflict with keywords of the rule file language because precedence rules always recognize keywords before names. User names cannot conflict with or contain symbols of the language (for example, {}). String constants can be used in place of names to remedy such a conflict, if it occurs.

This rule generally applies to complete keywords. For example, because there is a keyword “LVS Reduce Series Capacitors” does not mean that the word series is off-limits as a user-defined name. However, LVS is off-limits, because the syntax checker does not allow incomplete specification of a multi-word SVRF statement.

To minimize these restrictions, efforts have been made to allow keyword names and user-defined names to coincide when there is little or no possibility of any problem occurring.

Table 2-4 lists the keywords, keyword abbreviations, or prefixes of keywords that cause conflicts with user-defined names and result in compilation errors:

**Table 2-4. Keyword - Name Conflicts**

abut	drawn	internal	para	shift
acute	drc	intersecting	parallel	shrink
and	enc	label	parasitic	singular
angle	enclose	layer	path	size
angled	enclosure	layout	perimeter	snap
area	erc	length	perc	source
attach	exclude	litho	perp	sraf_fill
by	expand	lvs	perpendicular	space
capacitance	ext	magnify	pex	square
coin	extend	mask	pins	stamp
coincident	extent	measure	polygon	step
connect	extents	merge	port	svrf
connected	external	net	ports	tddrc
convex	flag	not	precision	text
copy	flatten	notch	proj	title
corner	fracture	obtuse	projecting	topex
cut	fringe	offgrid	push	touch
dbclassify	group	opcbias	rectangle	trace
deangle	grow	opclineend	rectangles	tvf
density	hcell	opcsbar	region	unit
dev	holes	opposite	resistance	variable
device	in	or	resolution	vertex
dfm	inductance	ornet	ret	virtual
direct	inside	out	rotate	with
disconnect	int	outside	sbar	xor
donut	interact	overlap	sconnect	

## Rules for Inlined Parameter Blocks

Inline code blocks define a set of parameters separate from the SVRF operation itself, yet still in the rule file. This also allows the parameters be defined as a separate, uniquely named block and referenced by a command in a rule file (or as a separate file).

For example, the [Litho File](#) statement specifies the File parameter of the Litho, Fracture, DBCLASSIFY, or MDPverify operation to be inlined as a separate block. The File parameter of an operation making a call to a Litho File may indicate a Litho File name:

```
Litho File setup.poly [
    iterations 5
    gridsize 0.00109375
    stepsize 0.00109375
    tilemicrons 80.000000
    sse OPC_MIN_EXTERNAL 0.075 0.075
]
Litho OPC poly File setup.poly
```

When this Litho operation is compiled, the rule file is searched for the Litho File specification statement name setup.poly. If found, then Litho File setup.poly is automatically used in the Litho operation. If not found, then setup.poly is assumed to be an external file.

An inlined code block uses syntax rules that differ from typical SVRF syntax (for example, different comment characters can be accepted in an inline block). [Table 2-1](#) lists the general syntactic conventions for SVRF rule files.

[Table 2-5](#) lists the special syntax rules for inline code:

**Table 2-5. Inlined Keyword/Command Syntax Rules**

Keyword/Command	Legal Comment Characters	Special Syntax Rules
DBCLASSIFY	// /*...*/	Backslashes (\) can be used for line continuations. Same syntax rules as an SVRF rule file.
Density Convolve	//	Keywords and parameters for the script may not appear on the same lines as the brackets.
DFM Spec Fill	//	Backslashes (\) can be used for line continuations. Same syntax rules as an SVRF rule file.
Fracture HITACHI Fracture JEOL Fracture MEBES Fracture MICRONIC Fracture NUFLARE Fracture VBOASIS	//	Keywords and parameters for the script may not appear on the same lines as the brackets.

**Table 2-5. Inlined Keyword/Command Syntax Rules (cont.)**

Keyword/Command	Legal Comment Characters	Special Syntax Rules
Litho DenseOPC Litho File Litho OPC Litho OPCverify Litho ORC Litho Printimage Litho PSMgate	//, #	<p>Backslashes (\) can be used for line continuations.</p> <p>Curly braces ({} ) can be used for the following purposes:</p> <ul style="list-style-type: none"> <li>• enclosing inline optical and resist models</li> <li>• declaring custom fragmentation in a fragmentLayer block</li> <li>• defining a loop in Tcl scripts</li> <li>• defining a denseopc_options block</li> </ul> <p>Double quotes in the Litho File block define strings in the Tcl portion of the Litho File block.</p> <p>Parentheses and square brackets in the Litho File block can be used for evaluating Tcl expressions.</p> <p>Using a forward slash (/) in an sse variable declaration is not allowed.</p> <p>If the C style block commenting convention /* */ is used anywhere in the Litho block, Calibre generates an error.</p>
MDP Embed	//	<p>Lines cannot be longer than 1023 characters (Calibre truncates any SVRF line to 1023 characters if you exceed this value).</p> <p>You cannot specify left- or right-brackets within the embedded SVRF statement group. Therefore, any occurrence of brackets must be replaced as follows: Left bracket ( [ ) is &lt;LB&gt; Right bracket ( ] ) is &lt;RB&gt;</p> <p>The left and right angle brackets (&lt; &gt;) for &lt;SVRFSTART&gt;, &lt;SVRFEND&gt;, &lt;LB&gt;, and &lt;RB&gt; are literal and required.</p>
MDP Checkmap MDP Mapsize MDP Maskopt	//	Keywords and parameters for the script may not appear on the same lines as the brackets.
MDPverify MDPmerge MDPstat	//	Keywords and parameters for the script may not appear on the same lines as the brackets. Argument text cannot contain either a left ([) or right bracket (]).

## Basic Rule File Structure

The statements and operations that drive the layout verification process are stored in a text file. You prepare this file, called the *rule file*, using any text editor. When the rule file is loaded, it is checked for syntax and other errors, and compiled. Because the compiler processes a rule file as a single entity, the order in which the elements are presented is generally not important. However, there are certain situations where elements are order-dependent. These are discussed in the descriptions of the operations in the chapter “[Summary of Rule File Elements](#)” on page 69, where applicable.

The following broad categories of elements are found in an SVRF rule file:

- Layer Assignments
- Global Layer Definitions
- Comments
- Include Statements
- Rule Check Statements {
  - Local Layer Definitions
  - Layer Operations
  - Rule Check Comments}
- Specification Statements
- Connectivity Extraction Operations
- Parasitic Extraction Statements
- Device Recognition Operations
- Conditionals
- Macros
- Runtime TVF Functions

These are discussed in detail next.

## Rule File Elements

The main categories of rule file elements appear in Table 2-6.

**Table 2-6. Rule File Elements**

Element	Description
Layer Assignments and Definitions	<p>Layer assignments are made with the <a href="#">Layer</a> specification statement. For example:</p> <pre>LAYER POLY 1 LAYER OXIDE 2</pre> <p>In these statements, layer numbers are assigned to layer names (usually called original or drawn layers). Layer name reassessments are not allowed.</p>

**Table 2-6. Rule File Elements (cont.)**

Element	Description
Layer Assignments and Definitions (cont.)	<p>Layer sets are layers with multiple layer numbers or layer names assigned to them. For example:</p> <pre>LAYER MET1 1 2 LAYER ALL_MET MET1 MET2 MET3</pre> <p>In general, layer sets behave like simple layers with few exceptions. These are discussed in Chapter 4 under the applicable statements.</p> <p>Layer definitions are a means for deriving layers from original layers. Once named, the derived layer can then be used in another operation in the rule file by referencing its name. A layer definition has the form:</p> $\text{name} = \textit{layer operation}$ <p>Layer definitions within a rule file can exist either inside or outside of a rule check statement. Layer definitions that are not inside of a rule check statement are said to be global. That is, the names of the layers that they define can be used in operations anywhere within the rule file. A layer definition that is inside of a rule check statement is said to be local to that rule check statement, that is, the layer name that it defines is not known outside of the rule check statement.</p> <p>In local scope, the layer names can be reused in different rule check statements. No two global layer definitions can share a name, nor can local layer definitions within the same rule check statement share a name. Any local layer definition will supersede a global definition of the same name within the rule check statement where it appears.</p>

**Table 2-6. Rule File Elements (cont.)**

<b>Element</b>	<b>Description</b>
Implicit Layer Definitions	<p>An implicit layer definition consists of a matching set of parentheses enclosing one layer operation. The layer operation may, in turn, have inputs that are implicitly derived. An implicit layer operation may be used as input to any operation.</p> <p><b>Example:</b></p> <pre>taps = (pdif AND (bulk NOT nwell)) OR (ndif AND nwell)</pre> <p>Here, there are two implicit layer definitions used as inputs to the OR operation. The first AND operation also has an implicit layer definition as input. Implicit layer definitions can be used in rule checks also.</p> <p><b>Example:</b></p> <pre>rule {EXT &lt; 0.1 (poly NOT oxide) oxide}</pre> <p>These types of definitions cannot be used in the <a href="#">Enclosure</a>, <a href="#">External</a>, and <a href="#">Internal</a> operations when the ( ) and [ ] derived edge layer operators are used.</p>
Comments	<p>The // characters start a rule file comment that terminates at the end of the line on which they occur. Multi-line C-style comments /* ... */ are also permitted.</p> <p>For comments that occur within a rule check (within the braces), the @ character is used. These comments occur in a rule check and can be displayed by various user interfaces like Calibre RVE.</p> <p>The ^ character may be used in a rule check comment to dereference the value of a variable in the comment. For example, if my_var is a numeric variable, then this syntax:</p> <pre>@ spacing is within ^my_var</pre> <p>causes the value of my_var to appear in the rule check output.</p>
Include Statements	<p>Inserts another rule file within a rule file. This statement behaves exactly as if all of the text of the rule file with the given file name were directly included in the rule file where the directive appears. The format is:</p> <pre>include <i>filename</i></pre> <p>In addition, multiple include statements can appear and the include depth is arbitrary. Recursive calling of included files is not allowed.</p>

**Table 2-6. Rule File Elements (cont.)**

<b>Element</b>	<b>Description</b>
Layer operations	<p>These are SVRF commands that manipulate layer data. They are often referred to simply as operations. This is in distinction to Specification Statements, which is the other major command category. See “<a href="#">Layer Operations</a>” on page 69.</p>
Rule Check Statements	<p>Particular to nmDRC-style applications, rule check statements specify layer operations whose resulting derived layers are placed into the results database when the rule check statement is executed by the application. They have the following basic form:</p> <pre><i>name {</i>   <i>@ rule check comment</i>   <i>layer operation or layer definition</i>   <i>...</i>   <i>layer operation or layer definition</i> <i>}</i></pre> <p>No two rule check statements in the rule file can share a name. Rule check statements do not need to span multiple lines nor contain comments. A rule check must have at least one standalone operation (that is, one operation that is <i>not</i> a layer definition of the form <i>x = layer operation</i>) that has output to the nmDRC or ERC results database or the rule check will not compile. Rule check names may not use special symbols unless the name is enclosed in quotation marks.</p>
Specification Statements	Specification statements serve to describe the overall environment for applications using the rule file. They establish input and output flow, as well as controlling how various algorithms perform their tasks. See “ <a href="#">Specification Statements</a> ” on page 84.
Connectivity Extraction Operations, Device Operation, and Parasitic Extraction Operations	These statements control connectivity extraction, device definitions, and parasitic extraction. These are discussed in the “ <a href="#">Summary of Rule File Elements</a> ” chapter.
Conditionals	Conditionals, or pre-processor directives, control how a rule file is compiled. They allow control over which parts of a rule file are used. See “ <a href="#">Pre-Processor Directives</a> .”
Macros	Macros are constructs that allow repeated use of SVRF operations in a type of template. Macros can be called multiple times in a rule file. See “ <a href="#">Macros</a> ” on page 64.
Runtime TVF Functions	Runtime TVF requires the presence of an SVRF element called TVF FUNCTION to define operations to perform during Calibre runtime. See “ <a href="#">Runtime TVF</a> ” on page 1906 for more information.

## Pre-Processor Directives

Pre-processor directives, also known as *conditionals*, come in the following categories.

- **Conditional Directives for Process Control** — These directives are structures that permit conditional compilation of rule file text. They allow you to include multiple processes, or variations of the same process, in one rule file.
- **Conditional Directive for Version Control** — This directive allows for easier integration of new functionality into existing SVRF files, by enabling conditional compilation based upon the Calibre version.
- **Conditional Directive for Environment Variables** — This directive allows specification of a default environment value that is used when an environment value is undefined.

Pre-processor directives have a number of special considerations:

- The pre-processor directives used for Calibre and ICverify applications are not as general as most C-language pre-processors. This results in testing only existence of variables, not expressions.
- You cannot put multiple pre-processor directives on one line.
- Each pre-processor directive must be the first entry on a line, except for spaces.
- Conditional compilation directives that have dependencies on Include statements or /\* ... \*/ comments are compiled after Include statements and before /\* ... \*/ comments.

## Conditional Directives for Process Control

The section of the rule file that Calibre or ICverify compiles can be controlled with the #DEFINE and #UNDEFINE keywords within your rule file. The syntax of these keywords follows:

```
#DEFINE name [value]
#UNDEFINE name
```

where **name** is a mandatory string and **value** is an optional string. You can also implicitly define **name** from the shell environment, in which case the #DEFINE or #UNDEFINE statements are not needed.

A **name** is considered *defined* if either of the following are true:

- Explicit definition: The **name** appears in a #DEFINE directive before it is referenced in a conditional, and the **name** does not appear in an #UNDEFINE statement prior to its reference in a conditional.
- Implicit definition: The **name** appears with a dollar character (\$) before it, and **name** is defined as a non-null shell environment variable.

You should enclose ***name*** within quotes if including spaces or characters other than letters and numbers.

The following is an example of an implicit definition of a ***name*** in the C shell environment:

```
setenv process 7lm
```

and the following is an example for a Bourne-style shell:

```
process=7lm; export process
```

The string “\$process” represents a ***name*** in an SVRF conditional statement (#IFDEF or #IFNDEF, which are discussed later). In these examples, the “\$process” variable would resolve to a value of “7lm”.

In an explicit ***name*** definition, the following construct establishes process as defined, with no assigned *value*:

```
#DEFINE process
```

and the following construct defines the ***name*** process and associates the *value* “7lm”:

```
#DEFINE process 7lm
```

This is analogous to the previous environment variable assignment examples, although “process” and “\$process” are not the same ***name***.

The *value* may not be specified as an environment variable in a #DEFINE directive. For instance, this is not allowed:

```
//in the shell, $process = 7lm, $value = 7lm
#DEFINE $process $value // variable not allowed for <value> parameter
```

In this example, \$value is treated as a literal string, not a variable.

If you define an environment variable (for example, setenv myvar xyz), such that the variable is used as an implicit ***name*** definition (\$myvar is implicitly defined for conditionals and resolves to xyz), and then you do the following:

```
#DEFINE $myvar newvalue
```

this causes \$myvar to resolve to newvalue, regardless of the shell environment. In other words, #DEFINE directives can override implicit ***name*** definitions from the shell environment. Note that #DEFINE statements do not affect the shell environment settings, only the rule file pre-processor.

To undefine a ***name***, the #UNDEFINE directive is used. Using the previous examples:

```
#UNDEFINE $process //undefines the implicit name definition
#UNDEFINE process //undefines the explicit name definition
```

Note, these do not undefine the same names. Note also that #UNDEFINE statements do not affect the shell environment, only the rule file pre-processor.

This conditional is controlled by the #IFDEF, #IFNDEF, #ELSE, and #ENDIF keywords, and uses the **name** and *value* strings to determine whether rule file text is executed. The form for this conditional is:

```
{ #IFDEF name [value] rule_file_text | #IFNDEF name [value] rule_file_text }
    [#ELSE rule_file_text]
    #ENDIF
```

where **name** is a string that is defined with the #DEFINE directive, or implicitly defined from the shell environment. The *value* is an optional string that appears in a #DEFINE directive, or it is the value that **name** is assigned as a variable in the shell environment. #IFDEF (acronym for “if defined”) and #IFNDEF (“if not defined”) test whether or not a **name** is defined, and they do the opposite of the other. #ELSE statements are executed if the preceding #IFDEF or #IFNDEF conditionals evaluate to “false”.

This syntax allows compilation control of #IFDEF and #IFNDEF statements, not only by a **name** definition, but also by a case-insensitive string comparison to the optional *value*. The string *rule\_file\_text* includes rule file statements or operations to be performed, depending on the pre-processor directive keywords. The *rule\_file\_text* can also contain conditionals, which creates nested pre-processor directives. (The [SVRF Error](#) statement is useful for inserting compiler error messages within conditional constructs.)

For the following discussion of conditionals, “associated *value*” refers to one of these cases:

- The *value* assigned to the **name** in the most recent #DEFINE directive, prior to the #IFDEF or #IFNDEF conditional, or
- The non-empty string, if any, assigned to **name** variable in the shell environment.

“#IFDEF **name** *value*” enables or disables conditional compilation as follows:

1. If **name** is not defined, then compilation is disabled.
2. Else if **name** is defined with a *value*, but *value* (*value* is optional) is not specified in the #IFDEF, then compilation is enabled.
3. Else if **name** is defined with no associated *value*, then compilation is disabled.
4. Else if *value* equals the assigned value of **name** (by case-insensitive string comparison), then compilation is enabled.
5. Else compilation is disabled.

“#IFNDEF **name** *value*” enables or disables conditional compilation as follows:

1. If **name** is not defined, then compilation is enabled.
2. Else if **name** is defined with a *value*, but *value* (*value* is optional) is not specified in the #IFNDEF, then compilation is disabled.
3. Else if **name** is defined with no associated *value*, then compilation is enabled.
4. Else if *value* equals the assigned value of **name** (by case-insensitive string comparison), and both are defined, then compilation is disabled.
5. Else compilation is enabled.

The way to change the state of a conditional compilation pre-processor directive is not to comment it out, but to change the directive to its opposite. For example, if you have this:

```
#DEFINE name
```

you should not precede this with a comment character because # must be the first character on the line for any pre-processor directive. The /\* ... \*/ comments will not work because they are processed after pre-processor directives when there is a dependency. So the proper way to negate the definition is to do this:

```
#UNDEFINE name
```

or this:

```
#DEFINE name
#UNDEFINE name //override the definition of name
...
```

### **Example 1**

In the Example 1, process P1 states that metal6 is the top metal layer, process P2 states that metal5 is the top metal layer, and in all other processes metal4 is the top metal layer. The arguments P1 and P2 would appear in #DEFINE or #UNDEFINE statements in a control file, which would include the following code:

```
LAYER metal4 23
LAYER metal5 26
LAYER metal6 14

#ifndef P1
    LAYER top_metal metal6
#else
    #ifdef P2
        LAYER top_metal metal5
    
```

```
#else
    LAYER top_metal metal4
#endif
#endif
```

### Example 2

In Example 2, ***name*** is initially defined for the conditionals, then it is undefined (order is important). P1 is explicitly defined:

```
#DEFINE P1
...           //rule file statements

#ifndef P1
...           //conditional statements are compiled
#endif

#ifndef P1
...           //conditional statements are not compiled
#endif

#UNDEFINE P1

#ifndef P1
...           //conditional statements are not compiled
#endif

#ifndef P1
...           //conditional statements are compiled
#endif
```

### Example 3

Example 3 shows use of ***name value*** pairs in an explicit definition. P1 is explicitly defined:

```
#DEFINE P1 71m //the associated value of P1 is "71m"

#ifndef P1
...           //conditional statements are compiled
#endif

#ifndef P1 71m
...           //conditional statements are compiled
...           //P1 = 71m from the #DEFINE directive
#endif

#ifndef P1 P1
...           //conditional statements are not compiled
...           //P1 = 71m from the #DEFINE directive
#endif

#ifndef P1
...           //conditional statements are not compiled
#endif
```

```
#IFNDEF P1 7lm
...
    //conditional statements are not compiled
    //P1 = 7lm from the #DEFINE directive
#endif

#IFNDEF P1 P1
...
    //conditional statements are compiled
#endif
```

#### **Example 4**

In Example 4, *name* is implicitly defined as a shell variable. Note the #UNDEFINE directive does not affect the environment variable declared in this example. “\$P1” is a non-null shell environment variable:

```
...      //rule file statements
#define $P1
...
    //conditional statements are compiled
    //P1 is defined in the shell
#endif

#ifndef $P1
...
    //conditional statements are not compiled
#endif

#UNDEFINE P1

#define $P1
...
    //conditional statements ARE compiled because
    //the previous #UNDEFINE does not affect $P1
#endif

#ifndef P1
...
    //conditional statements are compiled
    //P1 is currently undefined
#endif
```

#### **Example 5**

Example 5 shows use of name value pairs in an implicit definition. “\$P1” is a non-null shell environment variable assigned the value 71m:

```
#define $P1 71m
...
    //conditional statements are compiled
    //P1 = 71m in the shell environment
#endif

#ifndef $P1 71m
...
    //conditional statements are not compiled
#endif

#UNDEFINE P1
```

```
#IFDEF $P1
    //conditional statements are compiled
    //previous #UNDEFINE does not affect $P1
#endif

#define $P1 P1 // $P1 reassigned to P1

#ifndef $P1 P1
...
    //conditional statements are compiled
#endif
```

## Conditional Directive for Version Control

The version control preprocessor directive enables conditional compilation based upon the Calibre version, allowing for easier integration of new functionality into existing SVRF files. The form for this conditional is:

**#IFDEF CALIBRE\_VERSION *value***

where ***value*** is a valid Calibre version string of the form:

**v*DDDD.D\_D\*D\****

For example:

v2011.2\_37.32

This directive triggers compilation if the Calibre software version (displayed at the top of the banner in the Calibre transcript) is greater than or equal to the specified ***value***. “Greater than or equal to” means the Calibre software version must be the same or more advanced than the specified ***value***. If ***value*** is not specified, this directive is ignored.

### Example

In the following example, if the Calibre version is greater than or equal to v2011.2\_0.0, then perform the first OR operation. Otherwise, perform the second OR operation.

```
#IFDEF CALIBRE_VERSION v2011.2_0.0
    X = OR A B C D
else
    X = (A OR B) OR (C OR D)
#endif
```

## Conditional Directive for Environment Variables

The #PRAGMA preprocessor directive defines the default value for an environment variable used in SVRF statements. The syntax of the #PRAGMA statement is as follows:

**#PRAGMA ENV *variable\_name value***

Any SVRF operation or specification statement that is specified after the #PRAGMA ENV statement and refers to the environment variable *variable\_name* will use the value of the actual environment variable if one is defined; otherwise, the *value* from the #PRAGMA ENV statement is used.

The #PRAGMA ENV statement must be specified prior to the first use of the environment variable *variable\_name* in order to function.

Note that the #PRAGMA ENV statement applies only to the variables used in SVRF statements. It has no effect on the environment variables used by Tcl expressions in TVF code, including compile-time and runtime TVF.

#### **Example 1**

Assume that the environment variable INCLUDE\_DIR is not defined.

```
INCLUDE $INCLUDE_DIR/rules.inc
```

This statement does not compile.

#### **Example 2**

Assume that the environment variable INCLUDE\_DIR is defined as “./inc”.

```
INCLUDE $INCLUDE_DIR/rules.inc
```

This statement compiles and the Include statement will read the file “./inc/rules.inc”.

#### **Example 3**

Assume that the environment variable INCLUDE\_DIR is not defined.

```
#PRAGMA ENV INCLUDE_DIR "."
INCLUDE $INCLUDE_DIR/rules.inc
```

This statement compiles and the Include statement will read the file “./rules.inc”.

#### **Example 4**

Assume that the environment variable INCLUDE\_DIR is defined as “./inc”.

```
#PRAGMA ENV INCLUDE_DIR "."
INCLUDE $INCLUDE_DIR/rules.inc
```

This statement compiles and the Include statement will read the file “./inc/rules.inc”.

## Macros

Macros are similar to the macros of the C and C++ languages. Macros are functional templates that can be called multiple times in a rule file.

A macro definition consists of the keyword DMACRO (define macro), followed by a macro name, followed by a list of zero or more argument names, followed by “{”, followed by a sequence of zero or more SVRF statements or operations, followed by “}”. This is the syntax:

```
DMACRO macro_name [arguments] {  
    SVRF_code }
```

For example:

```
DMACRO WIDTH_CHECK lay val {  
    S1 = INT lay < val ABUT < 90 SINGULAR REGION  
    S2 = INT lay < val ANGLED == 2 PARALLEL OPPOSITE REGION  
    S3 = INT lay < val CORNER TO EDGE REGION  
    ( S1 OR S2 ) OR S3  
}
```

DMACRO names must be unique, each argument must be a name, and an argument may not be duplicated in the same DMACRO argument list. DMACRO definitions may not be nested.

A macro is invoked by the keyword CMACRO (call macro) followed by a macro name and a list of zero or more arguments:

```
CMACRO macro_name [arguments]
```

The *macro\_name* must match that of some DMACRO definition. Each argument may be either a name (generally a layer or variable name) or a numeric constant. A sufficient number of *arguments* must be present after the CMACRO *macro\_name* to match what the DMACRO specifies as arguments.

The DMACRO definition within its braces is internally placed in the CMACRO call, with argument substitution. For example, these CMACRO calls refer to the previously-shown DMACRO named WIDTH\_CHECK:

```
poly_width { CMACRO WIDTH_CHECK poly 0.5 }  
metal_width { CMACRO WIDTH_CHECK metal 0.6 }
```

The arguments poly and 0.5 are substituted into the DMACRO named WIDTH\_CHECK, and the DMACRO definition, with argument substitution, becomes the rule check poly\_width. Similarly, this occurs for metal\_width.

You can also do this:

```
width_checks {  
    CMACRO WIDTH_CHECK poly 0.5  
    CMACRO WIDTH_CHECK metal 0.6  
}
```

Layer definition names in DMACRO statements are locally-scoped by default when the DMACRO is instantiated in a CMACRO. This is done by generating a unique (rule-file-wide) name for each layer definition in the DMACRO construct when the DMACRO is instantiated. References to the layer definition name in the DMACRO are also substituted appropriately. This is illustrated by the previously shown width\_checks example. The compiler is not confused by multiple intermediate derived layers having the same names (S1, S2, S3, from the DMACRO WIDTH\_CHECK) in the width\_checks rule.

If an argument to a DMACRO statement matches the name of a derived layer within the DMACRO block definition, then the corresponding CMACRO argument becomes the name of a global layer. For example:

```
DMACRO MAC IN1 IN2 OUT { // OUT occurs as an argument to DMACRO
  X = IN1 AND IN2
  OUT = SIZE X BY 0.01      // OUT is a derived layer name
}
CMACRO MAC L1 L2 RES      // OUT is assigned the global layer name RES
```

In this example, OUT is used as a DMACRO argument name and is the name of a derived layer in the DMACRO; therefore, OUT is treated as a global name for the derived layer. The global name OUT becomes the global layer name RES because the CMACRO argument called RES corresponds to OUT.

**Variable** statements inside of a DMACRO are locally-scoped and are not accessible outside of the DMACRO. This enables you to use the same variable name with different values inside multiple DMACRO definitions. Variable statements that are specified outside of a DMACRO are globally-scoped and available for any DMACRO in the rule file. If present, a locally-scoped variable will override a globally-scoped variable of the same name.

The LOCAL() modifier can be used to create local names that are valid only in the scope of each CMACRO call. This modifier is useful for defining specification statements within DMACROs. Consider the following example:

```
DMACRO COPY_C L1 L2 C {
  LAYOUT CELL LIST LIST1 C
  L2 = DFM COPY L1 CELL LIST LIST1
}
```

Assume that you want to use this DMACRO as follows:

```
LAYER A 1
LAYER B 2
CMACRO COPY_C A A1 "A*"
CMACRO COPY_C B B1 "B*"
```

This use is invalid because the [Layout Cell List](#) specification statement is duplicated with the name LIST1. To resolve this problem, a LOCAL() modifier can be added to the DMACRO:

```
DMACRO COPY_C L1 L2 C {
    LAYOUT CELL LIST "LOCAL(LIST1)" C
    L2 = DFM COPY L1 CELL LIST "LOCAL(LIST1)"
}
CMACRO COPY_C A A1 "A*"
CMACRO COPY_C B B1 "B*"
```

The name “LOCAL(LIST1)” is replaced by a name that is unique to each CMACRO call. Within one CMACRO call, multiple instances of LOCAL(name) are always replaced by the same local name (such as “LOCAL(LIST1)” in the example above). Different names used with LOCAL() in the same macro remain different after the replacement. Note that the quotes around the string “LOCAL(LIST1)” are required.

DMACRO definitions may themselves contain nested CMACROs. However, no CMACRO can call a DMACRO that contains a call to that CMACRO (recursion is not allowed). For example, this is allowed:

```
DMACRO skew_edge inlayer {
    ANGLE inlayer > 0 < 45
    ANGLE inlayer > 45 < 90
}

DMACRO acute_angle inlayer grid_size {
    INT inlayer < 4*grid_size ABUT > 0 < 90 INTERSECTING ONLY
    EXT inlayer < 4*grid_size ABUT > 0 < 90 INTERSECTING ONLY
}

DMACRO illegal_shape met grid_size {

    // nested CMACROs
    CMACRO skew_edge met
    CMACRO acute_angle met grid_size
}

test {CMACRO illegal_shape metall .001}
```

Any of the rule file comment characters may be used in a Macro definition; however, user comments (specified by the @ symbol) will not appear in the nmDRC results database (or in Calibre RVE). User comments should be placed in the rule check that calls the DMACRO.

**Macros in user-defined Device property calculation** — You can use macros for user-defined property calculation in [Device](#) operations.

Argument substitution is performed for arguments that appear within square brackets ([ ]) in the DMACRO definition. Note the square brackets must appear in the DMACRO definition.

For example:

```
DMACRO mosprop seed {
[
    property W, L
    weffect = 0 // Specify width effect constant here
                // (built-in default is 0).
```

```

W = 0.5 * ( perim_co( S, seed ) + perim_in( S, seed )
             + perim_co( D, seed ) + perim_in( D, seed ) )
L = area( seed ) / W
    if ( ( bends( seed ) != 0 ) && ( weffect != 0 ) ) {
        if( W > L ) W = W - weffect * bends( seed ) * L
        else L = L - weffect * bends( seed ) * W
    }
}
]

DEVICE MP gate gate psd psd CMACRO mosprop gate
DEVICE MN gate gate nsd nsd CMACRO mosprop gate

```

The gate layer in the CMACRO call in each Device statement gets passed in as the seed argument in the mosprop macro.

CMACRO calls within square brackets are not expanded. For example, this is not valid:

```

DEVICE MP gate gate psd psd [ CMACRO MOSPROP2 gate ] // Bad.
DEVICE MN gate gate nsd nsd [ CMACRO MOSPROP2 gate ] // Bad.

```



# Chapter 3

## Summary of Rule File Elements

---

This chapter provides a summary of information on the following elements used in preparing rule files:

- [Layer Operations](#)
- [Connectivity Extraction Statements](#)
- [Device Recognition Statement](#)
- [Fracturing Statements](#)
- [Post-Design Correction Statements](#)
- [Specification Statements](#)

For detailed information concerning syntax, options and parameters, functional description, or usage examples for a particular statement or operation, refer to the applicable subsection of the “[Reference Dictionary](#).”

## Layer Operations

Layer operations are functions which, when evaluated, create a derived layer from other derived layers or original layers. There are two types of layer operations: dimensional check operations and auxiliary operations. The output of these operations falls into three categories: error-directed, edge-directed, and polygon-directed. These are discussed in detail in the “nmDRC Concepts” chapter of the [Calibre Verification User’s Manual](#).

Error-directed output consists of output clusters of edge segments; this output is sent to the nmDRC results database and may not be used to form derived layers. The dimensional check operations discussed later in this chapter are primarily responsible for this type of output; however, [Drawn Acute](#), [Drawn Angled](#), [Drawn Skew](#), [Drawn Offgrid](#), and [Offgrid](#) are examples of auxiliary operations that also produce this type of output.

Edge-directed output is output in the form of edge segments. It can be sent to the nmDRC results database and can also be used for derived layer generation.

Polygon-directed output is output in the form of polygons. It can be sent to the nmDRC results database and can also be used for derived layer generation.

All operations also fall into two sub-categories: layer selectors and layer constructors. Layer selectors select data from appropriate input layers based upon certain properties that the corresponding operations search for. Layer constructors construct new polygon data based upon

characteristics of the input layers. The category to which these operations belongs is annotated in the tables that follow.

Layer selectors have an additional interesting property in that they are node-preserving (or net-preserving); that is, they pass connectivity information. They do it in the following manner.

Given:

$x = \text{layer1 } operation$   
 $y = \text{layer1 } operation \text{ layer2}$

In each of these cases, connectivity information (that is, net IDs) is passed from layer1 to the output layer. A limited set of layer constructors also have this property, like AND and NOT. Refer to the “[Node-Preserving Layer Operations](#)” section of the *Calibre Verification User’s Manual* for further discussion.

Layer of origin is an important consideration when dealing with derived layers and connectivity information. Refer to the “[Layer of Origin](#)” section of the *Calibre Verification User’s Manual* for a complete discussion.

---

**Note**



Throughout the tables in this section, layer constructors are denoted with a superscript <sup>c</sup>,  
selectors with a superscript <sup>s</sup>, and <sup>cs</sup> if the operation can fulfill either role.

---

## Dimensional Check Layer Operations

Table [3-1](#) describes briefly the three dimensional check operations. The secondary keywords and parameters used to modify the default behavior of these operations are described with the applicable operation in the “[Reference Dictionary](#)” section. These operations can be used in all Calibre and ICVerify applications.

These operations are error-directed, edge-directed, or polygon-directed, depending on the secondary keywords used. When these operations are configured as polygon-directed, they act as layer constructors. When they are configured as error-directed (which they are by default) or edge-directed (through the use of the [ ] and ( ) layer operators), they are layer selectors. When they are configured as edge-directed, they are also node-preserving. Here are some examples:

```
rule1 { ENC contact metall1 < .06 }
// error-directed output to nmDRC results database

rule2 { x = ENC [contact] metall1 < .06
        COPY x }
// edge-directed Enclosure output to derived layer x

rule3 { ENC contact metall1 < .06 REGION }
// polygon-directed output to nmDRC results database
```

**Table 3-1. Layer Operations: Dimensional Check Operations**

Operation	Description
Enclosure <sup>cs</sup>	Measures the separation between the <i>exterior</i> sides of edges from layer1 and the <i>interior</i> sides of edges from layer2. Measured edge pairs that satisfy the given constraint are output. Intersecting edge pairs are not measured by default. This operation is not commutative; therefore, the order of the input layers is significant. This operation is polygon-directed if you use the REGION keyword. This operation is edge-directed if you use the [ ] and ( ) operators. Otherwise, it is error-directed.
External <sup>cs</sup>	Measures the separation between the <i>exterior</i> sides of edges from the input layer or from each of the two input layers. Measured edge pairs that satisfy the given constraint are output. Intersecting edge pairs are not measured by default. This operation is polygon-directed if you use the REGION keyword. This operation is edge-directed if you use the [ ] and ( ) operators. Otherwise, it is error-directed.
Internal <sup>cs</sup>	Measures the separation between the <i>interior</i> sides of edges from the input layer or from each of the two input layers. (In the one-layer case, measured edges must come from the same polygon). Measured edge pairs that satisfy the given constraint are output. Intersecting edge pairs are not measured by default. This operation is polygon-directed if you use the REGION keyword. This operation is edge-directed if you use the [ ] and ( ) operators. Otherwise, it is error-directed.
Rectangle Enclosure <sup>c</sup>	Creates a derived polygon layer consisting of rectangles from one layer enclosed by another layer. Intended for complex enclosure checks.
TDDRC <sup>c</sup>	Layer operation for performing relatively simple table-based nmDRC checks. This operation is polygon-directed if you use the REGION keyword. This operation is edge-directed if you use the [ ] operator. Used only in Calibre.

<sup>c</sup> denotes layer constructor

<sup>s</sup> denotes layer selector (also node-preserving)

<sup>cs</sup> denotes operation can fulfill either function

## Auxiliary Layer Operations

Table 3-2 lists the auxiliary operations. These operations are grouped under the categories: polygon-directed auxiliary operations, edge-directed auxiliary operations, and error-directed auxiliary operations. For detailed descriptions of these operations, refer to the applicable “Reference Dictionary” subsection.

The operations that appear in parentheses as (Not *operation*) reflect the syntax of the logical negation of the corresponding statement in the table cell. The behaviors that occur in parentheses in the Description cells of the table reflect what these logical negations do. These operations can be used by all Calibre and ICverify applications except where noted.

**Table 3-2. Layer Operations: Auxiliary Operations**

Operation	Description
<b>Polygon-directed Auxiliary Operations (derive polygon layers):</b>	
<a href="#">AND<sup>c</sup></a>	One- or two-layer Boolean operation that selects the polygon area from the input layer covered by multiple polygonal areas. A one-layer operation can select a specified number of polygon areas common to more than one polygon. The two-layer form is node-preserving.
<a href="#">Area<sup>s</sup> (Not Area)<sup>s</sup></a>	Selects polygons from the input layer that (do not) have a specified area.
<a href="#">Cut<sup>s</sup> (Not Cut)<sup>s</sup></a>	Selects all layer1 polygons that (do not) share some, but not all of their area with layer2 polygons.
<a href="#">Deangle<sup>c</sup></a>	Creates a derived polygon layer by replacing skew (not orthogonal or at a 45-degree angle with respect to the database axes) edges with sequences of orthogonal or 45-degree edges.
<a href="#">Density<sup>c</sup></a>	Generates a derived polygon layer by measuring the density of its input layer over a user-specified area within a user-specified window.
<a href="#">Density Convolve<sup>c</sup></a>	Creates a representation of the input layer that shows density values after applying convolution algorithms.
<a href="#">Device Layer<sup>c</sup></a>	Constructs a layer that contains seed shapes from all device instances with the specified and model_name. Seed shapes are those from <i>device_layer</i> arguments in Device operations. Used only in LVS applications.
<a href="#">DFM Analyze<sup>c</sup></a>	Layer operation for analysis that is used to improve manufacturing yield. Specifically, it provides output of layout regions that meet specified constraints for various layout properties. Requires a Calibre YieldAnalyzer product license.
<a href="#">DFM CAF</a>	Calculates a numeric critical area value on one or more input layers.
<a href="#">DFM Connectivity Redundant<sup>s</sup></a>	Outputs a layer that contains redundant or non-redundant polygons.
<a href="#">DFM Copy<sup>s</sup></a>	Copies layers of any type to a new layer, and allows selection of specific cells to copy.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

<b>Operation</b>	<b>Description</b>
<a href="#">DFM Create Layer<sup>c</sup></a>	Used only within a YieldServer script. Creates a new layer in a DFM Database.
<a href="#">DFM Critical Area<sup>c</sup></a>	Layer operation used to identify yield limiting conditions. Requires a Calibre YieldAnalyzer product license.
<a href="#">DFM Expand Edge<sup>c</sup></a>	Expands edges based on DFM properties attached to the input layer.
<a href="#">DFM Expand Enclosure<sup>c</sup></a>	Generates regions of expanded enclosure for vias. Requires a Calibre YieldEnhancer product license.
<a href="#">DFM Fill<sup>c</sup></a>	Creates a derived polygon layer containing fill polygons generated according to a fill specification. Requires a Calibre YieldEnhancer product license.
<a href="#">DFM GCA</a>	Outputs a derived polygon layer that represents areas susceptible to producing open or short circuits in the presence of random particles of a given radius.
<a href="#">DFM Grow<sup>c</sup></a>	Expands polygons uniformly. Requires a Calibre YieldEnhancer product license.
<a href="#">DFM Histogram</a>	Analyzes the distribution of DFM Property values and computes a histogram.
<a href="#">DFM Measure<sup>c</sup></a>	Performs single-layer, table-based width and spacing checks. This operation is polygon directed if you use the REGION keyword. You can optionally output the results to a results database (RDB) for statistical analysis. Requires a Calibre YieldAnalyzer product license.
<a href="#">DFM Property<sup>c</sup></a>	Associates user-defined properties to individual objects on the input layer and creates a derived layer containing only those objects that meet the specified criteria
<a href="#">DFM RDB<sup>c</sup></a>	Writes original or derived layer data to an ASCII RDB (results database). While technically a layer constructor operation, the derived layer created <i>in memory</i> by this operation is always empty.
<a href="#">DFM Redundant Vias<sup>s</sup></a>	Identifies redundant or non-redundant polygons on a layout and produces a derived polygon layer containing the identified polygons. Used with <a href="#">DFM Spec Via Redundancy</a> .
<a href="#">DFM Shift Edge<sup>c</sup></a>	Shifts edges based on DFM properties attached to the input layer.
<a href="#">DFM Size<sup>c</sup></a>	Oversizes or undersizes the polygons on the input layer by the amount specified.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

Operation	Description
DFM Transform <sup>c</sup>	Transforms geometries on an input layer.
DFM Transition <sup>c</sup>	Adds metal and vias to interconnect transitions. Requires a Calibre YieldEnhancer product license.
Donut <sup>s</sup> (Not Donut) <sup>s</sup>	Measurement operation that selects polygons from the input layer that (do not) have interior cycles.
Enclose <sup>s</sup> (Not Enclose) <sup>s</sup>	Selects all layer1 polygons that (do not) completely enclose any layer2 polygon.
Enclose Rectangle <sup>s</sup> (Not Enclose Rectangle) <sup>s</sup>	Selects all polygons on the specified layer that can (cannot) enclose a rectangle of the specified dimensions.
Expand Edge <sup>c</sup>	Generates a derived polygon layer by converting edges on the input layer into rectangles according to the expansion parameters.
Expand Text <sup>c</sup>	Creates a derived polygon layer consisting of merged squares centered on the positions of text objects having the supplied text name.
Extent <sup>c</sup>	Generates a derived polygon layer consisting of one rectangle that equals the database extent read in at runtime, or that represents the minimum bounding box of all shapes on the input layer.
Extent Cell <sup>c</sup>	Generates a derived polygon layer consisting of rectangles that represent the extents of cells in the given list. Used only in Calibre.
Extent Drawn <sup>c</sup>	Generates a derived polygon layer consisting of the extent of shapes (not text) on the specified input layers.
Extents <sup>c</sup>	Generates a derived polygon layer consisting of the merged, minimum bounding boxes of each polygon on layer.
Grow <sup>c</sup>	Generates a derived polygon layer where outward expansion of polygons from the input layer occurs in directions you specify.
Holes <sup>c</sup>	Generates a derived polygon layer by constructing polygons that would exactly fit inside of holes in polygons from the input layer.
Inside <sup>s</sup> (Not Inside) <sup>s</sup>	Selects all layer1 polygons that (do not) share all of their area with a layer2 polygon.
Inside Cell <sup>c</sup> (Not Inside Cell) <sup>c</sup>	Selects all polygons from the input layer that are (not) inside the sub-hierarchy of the specified cells. Used only in Calibre.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

<b>Operation</b>	<b>Description</b>
<b>Interact<sup>s</sup> (Not Interact)<sup>s</sup></b>	Selects all layer1 polygons that (do not) either share some or all of their area with a layer2 polygon, or are outside all layer2 polygons, but share a coincident edge or portion thereof with some layer2 polygon.
<b>Litho DenseOPC<sup>c</sup></b>	Specifies to run Calibre nmOPC during Calibre nmDRC or nmDRC-H processing and to place the results on a layer you specify.
<b>Litho OPC<sup>c</sup></b>	Specifies to run Calibre OPCpro during Calibre nmDRC or nmDRC-H processing and to place the results on a layer you specify.
<b>Litho OPCverify<sup>c</sup></b>	Specifies to run OPCverify during Calibre nmDRC or nmDRC-H processing and to place the results on a layer you specify.
<b>Litho ORC<sup>c</sup></b>	Specifies to run ORC during Calibre nmDRC or nmDRC-H processing and to place the results on a layer you specify.
<b>Litho Printimage<sup>c</sup></b>	Specifies to run Calibre PRINTimage during Calibre nmDRC or nmDRC-H processing and to place the results on a layer you specify.
<b>Litho PSMgate<sup>c</sup></b>	Specifies to run Calibre PSMgate during Calibre nmDRC or nmDRC-H processing and to place the results on a layer you specify.
<b>Magnify<sup>c</sup></b>	Magnifies the polygons on the input layer by the amount specified in the numeric parameter.
<b>Net<sup>s</sup> (Not Net)<sup>s</sup></b>	Connectivity-related operation that selects all layer1 polygons that are (not) on the specified nets.
<b>Net Area<sup>s</sup></b>	Connectivity-related operation that selects all layer1 polygons that lie on an electrical node such that the total area of polygons on layer1, on the same node, satisfies the given constraint.
<b>Net Area Ratio<sup>cs</sup></b>	Connectivity-related operation that selects all polygons from layer1 that lie on an electrical node and produce an expression value that meet the constraint. This operation can be either a constructor or a selector, depending on the options used.
<b>Net Area Ratio Accumulate<sup>c</sup></b>	Connectivity-related operation that outputs polygons from layer1 when the constraint is satisfied. The value of the operation's expression is attached to the output layer.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

Operation	Description
Net Interact <sup>c</sup>	Connectivity-related operation that outputs polygons from layer1 that interact with the specified number of nodes associated with polygons from layer2.
NOT <sup>c</sup>	Two-layer Boolean operation that selects all layer1 polygon areas not common to layer2 polygons. Node-preserving layer constructor.
Opcbias <sup>c</sup>	Calibre RET operation that performs biasing for dense line and via configurations. Requires a Calibre OPCpro or Calibre TDopc product license.
Opclineend <sup>c</sup>	Calibre RET operation that performs variable line-end extension, from simple end extension to hammerhead creation. Requires a Calibre OPCpro or Calibre TDopc product license.
Opctsbar <sup>c</sup>	Calibre RET operation that creates scattering bars for optical correction. Requires a Calibre OPCTSbar product license.
OR <sup>c</sup>	One- or two-layer Boolean operation that merges all polygons into one polygon. For two-layer operations, OR finds the union of polygons on input layers. The one-layer version is rarely useful because all applications using SVRF pre-merge data by layer.
Ornet <sup>c</sup>	Connectivity-related operation that combines (without fully merging) all overlapping layer1 and layer2 polygons on the same net. Node-preserving layer constructor. Performed flat in hierarchical applications.
Outside <sup>s</sup> (Not Outside) <sup>s</sup>	Selects all layer1 polygons that (do not) lie completely outside layer2 polygons.
Pathchk <sup>c</sup>	Constructs layers consisting of non-node-preserving nets in the layout that do or do not have a path to power, ground, or labeled nets, or that satisfy a combination of these conditions. Used only in LVS applications for ERC checks.
Perimeter <sup>s</sup>	Selects polygons from the input layer having a specified perimeter.
Pins <sup>c</sup>	ICVerify operation that generates a derived polygon layer consisting of all pin shapes on the specified original layer.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

Operation	Description
Ports <sup>c</sup>	ICVerify operation that generates a derived polygon layer consisting of all port shapes on the specified original layer. The port shapes are selected (only) from the top-level cell template.
Push <sup>c</sup>	For hierarchical applications, the input layer is pushed to the lowest possible level (that is, away from the top-level cell) of the hierarchy. In flat applications, performs like Copy.
Rectangle <sup>s</sup> (Not Rectangle) <sup>s</sup>	Selects all layer1 polygons that are (not) rectangles, with optionally specified dimensions.
Rectangles <sup>c</sup>	Methodology operation that generates a derived polygon layer consisting of arrays of rectangles with specified dimensions and spacing. Used for planarization.
RET NMDPC	Calibre RET operation used to decompose a layer into two masks. Requires a Calibre Double Patterning product license.
RET PXOPC	Calibre RET operation that performs pixel-based OPC and optimized SRAF insertion. Requires a Calibre pxOPC product license.
RET SBAR	Calibre RET operation that takes a target layer and uses the geometries on that layer and the templates specified in the litho file to output a new layer of SRAFs.
RET Sraf_Fill	Calibre RET operation that eliminates holes in SRAF layers generated by the OPCsbar or cnsraf commands. Requires a Calibre nmSRAF product license.
Rotate <sup>c</sup>	Creates a derived polygon layer by rotating all polygons on the input by layer by the given angle. Performed flat in hierarchical applications.
Shift <sup>c</sup>	Relocates polygons on the input layer by translating them in the x- and y-directions.
Shrink <sup>c</sup>	Contracts polygons inwardly in the specified directions.
Size <sup>c</sup>	Oversizes or undersizes the polygons on the input layer by the amount specified.
Snap <sup>c</sup>	Snaps each vertex on the input layer to the grid specified by the resolution parameter.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

Operation	Description
<b>Stamp<sup>c</sup></b>	Connectivity-related operation that creates a layer containing all layer1 polygons that are overlapped by layer2 polygons, and transfers the nodal information from the layer2 overlapping polygon to the overlapped layer1 polygon.
<b>Topex<sup>c</sup></b>	ICverify operation that generates a derived polygon layer consisting of all shapes on the specified original layer that are from the top-level cell template and have external aspect.
<b>Touch<sup>s</sup> (Not Touch)<sup>s</sup></b>	Selects all layer1 polygons that are (not) outside of all layer2 polygons and (do not) share a coincident edge or portion thereof with a layer2 polygon.
<b>Vertex<sup>s</sup></b>	Selects all layer1 polygons having a vertex count or that satisfies the given constraint.
<b>With Edge<sup>s</sup> (Not With Edge)<sup>s</sup></b>	Selects all layer1 polygons that (do not) have an edge or edge segment on layer2.
<b>With Neighbor<sup>s</sup> (Not With Neighbor)<sup>s</sup></b>	Selects all layer1 orthogonal rectangles that (do not) have the specified number of orthogonal rectangles within the specified distance.
<b>With Text<sup>s</sup> (Not With Text)<sup>s</sup></b>	Selects all layer1 polygons that (do not) intersect the position of the specified text object.
<b>With Width<sup>c</sup> (Not With Width)<sup>c</sup></b>	Selects all polygons (not) having the specified width.
<b>XOR<sup>c</sup></b>	One- or two-layer Boolean operation that creates a derived polygon layer containing all polygon regions contained by exactly one polygon.
<b>Edge-directed Auxiliary Operations (derive edge layers):</b>	
<b>Angle<sup>s</sup> (Not Angle)<sup>s</sup></b>	Selects all layer1 edges whose smaller angle magnitude with the x-axis of the coordinate system (does not) satisfy the given constraint.
<b>Coincident Edge<sup>s</sup> (Not Coincident Edge)<sup>s</sup></b>	Selects all layer1 edges or portions of edges that (do not) coincide with an edge from layer2.
<b>Coincident Inside Edge<sup>s</sup> (Not Coincident Inside Edge)<sup>s</sup></b>	Selects all layer1 edges or portions of edges that (do not) coincide with an edge from layer2 having the same interior side.
<b>Coincident Outside Edge<sup>s</sup> (Not Coincident Outside Edge)<sup>s</sup></b>	Selects all layer1 edges or portions of edges that (do not) coincide with an edge from layer2 having the opposite interior side.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

Operation	Description
<a href="#">Convex Edge<sup>s</sup></a>	Creates a derived edge layer by selecting edges dependent on number of convex endpoints, edge length, angle of abutting edge, and length of abutting edge.
<a href="#">DFM Measure<sup>c</sup></a>	Layer operation used to perform single-layer, table-based width and spacing checks. This operation is edge-directed if you use the [ ] operators. You can optionally output the results to a results database (RDB) for statistical analysis. Requires a Calibre YieldAnalyzer product license.
<a href="#">Inside Edge<sup>s</sup> (Not Inside Edge)<sup>s</sup></a>	Selects all layer1 edges or portions of edges that are (not) completely inside of a layer2 polygon.
<a href="#">Length<sup>s</sup> (Not Length)<sup>s</sup></a>	Selects all layer1 edges having length that (do not) satisfy the given constraint.
<a href="#">OR Edge<sup>c</sup></a>	Boolean operation that merges two derived edge input layers.
<a href="#">Outside Edge<sup>s</sup> (Not Outside Edge)<sup>s</sup></a>	Selects all layer1 edges or portions of edges that are (not) completely outside of a layer2 polygon.
<a href="#">Path Length<sup>s</sup></a>	Selects all edges on a maximal continuous path of edges from a single polygon having a total length that satisfies the given constraint.
<a href="#">Touch Edge<sup>s</sup> (Not Touch Edge)<sup>s</sup></a>	Selects all complete layer1 edges that (do not) coincide with an edge or segment from layer2.
<a href="#">Touch Inside Edge<sup>s</sup> (Not Touch Inside Edge)<sup>s</sup></a>	Selects all complete layer1 edges that (do not) coincide with an edge or segment from layer2 having the same interior side.
<a href="#">Touch Outside Edge<sup>s</sup> (Not Touch Outside Edge)<sup>s</sup></a>	Selects all complete layer1 edges that (do not) coincide with an edge from layer2 on the exterior side.
<b>Error-directed Auxiliary Operations (derive error layers, non node-preserving):</b>	
<a href="#">Drawn Acute<sup>s</sup></a>	Selects acute angles on any original geometry read from the layout database.
<a href="#">Drawn Angled<sup>s</sup></a>	Selects edges that are not orthogonal to the database axes on any original data read from the layout database.
<a href="#">Drawn Offgrid<sup>s</sup></a>	Selects off-grid vertices on any original geometry read from the layout database.
<a href="#">Drawn Skew<sup>s</sup></a>	Selects skew edges (edges which are non-orthogonal and do not have a slope of $\pm 1$ ) on any original geometry read from the layout database.

**Table 3-2. Layer Operations: Auxiliary Operations (cont.)**

Operation	Description
Offgrid <sup>s</sup>	Selects off-grid vertices on the input layer. Offgrid is error-directed by default, but can be edge-directed or polygon directed, depending upon the selected options.
<b>Miscellaneous Operations:</b>	
Copy <sup>s</sup>	Copies the input layer to the output layer.
Flatten <sup>c</sup>	For ICrules and Calibre nmDRC, functionally equivalent to Copy. For Calibre nmDRC-H, constructs a flat copy of the input layer.
Inductance MICheck	Selects geometries whose estimated mutual inductance violates a constraint.
Merge <sup>c</sup>	For ICrules and Calibre nmDRC, functionally equivalent to Copy. For Calibre nmDRC-H, the resulting layer is a merged copy of the input layer.
TVF <sup>c</sup>	Generates a single output layer based upon one or more input layers. The TVF operation serves as a call to a TVF FUNCTION and is used in the runtime TVF environment.

<sup>c</sup> denotes layer constructor

<sup>s</sup> denotes layer selector (also node-preserving except for error-directed operations)

<sup>cs</sup> denotes operation can fulfill either function

The TVF FUNCTION element serves as a macro for runtime layer manipulation. It is used in conjunction with the TVF layer operation. See “[Runtime TVF](#)” on page 1906.

## Connectivity Extraction Statements

Connectivity extraction statements define the electrical connectivity of a chip. They support many varied applications including on-the-fly connectivity maintenance within ICVerify, connectivity-based dimensional check operations within the nmDRC system, parasitic extraction operations, and LVS.

Table 3-3 lists the connectivity extraction operations. For detailed descriptions of these statements, refer to the applicable “[Reference Dictionary](#)” subsection.

**Table 3-3. Connectivity Extraction Operations**

Operation	Description
Attach	Transfers connectivity information from a layer1 object to a layer2 object. Used for label attachment.
Connect	Specifies electrical connection between objects on input layers.

**Table 3-3. Connectivity Extraction Operations (cont.)**

Operation	Description
<a href="#">Disconnect</a>	Allows the total deletion of existing connectivity model. Applies only to nmDRC applications.
<a href="#">Label Order</a>	Determines the order in which connectivity extraction looks for objects (typically shapes and paths) that intersect another object owning a label.
<a href="#">Sconnect</a>	Specifies one-directional connection between objects on specified layers. Used in detection of soft connections.
<a href="#">Stamp</a>	Connectivity-related layer operation that creates a layer containing all layer1 polygons that are overlapped by layer2 polygons, and transfers the nodal information from the layer2 overlapping polygon to the overlapped layer1 polygon.

## Virtual Connect Specification Statements

Table 3-4 lists the Virtual Connect specification statements. These form a single net from two or more (disjoint) nets that either share the same (case insensitive) name, or a portion of a name when a colon (:) or semicolon (;) is present. Used by all Calibre and ICverify applications except where noted.

**Table 3-4. Specification Statements: Virtual Connect**

Statement	Description
<a href="#">Virtual Connect Box Colon</a>	Specifies virtual connections for identical net names containing a colon (:) in box cells. Used in Calibre nmLVS-H and Calibre xRC only.
<a href="#">Virtual Connect Box Name</a>	Specifies virtual connections for identical (case insensitive) net names in box cells. Used in Calibre nmLVS-H and Calibre xRC only.
<a href="#">Virtual Connect Colon</a>	Specifies identical net names containing a colon (:) to be virtually connected.
<a href="#">Virtual Connect Depth</a>	Specifies the hierarchical depth at which virtual connections are made.
<a href="#">Virtual Connect Incremental</a>	Allows incremental virtual connections within all Connect blocks in Calibre nmDRC-H (hierarchical) flows.
<a href="#">Virtual Connect Name</a>	Specifies identical net names (case insensitive) to be virtually connected.
<a href="#">Virtual Connect Report</a>	Directs all Calibre and ICverify applications to report a warning (nmDRC applications) or a note (other applications) when a virtual connection is made by the connectivity extractor.

**Table 3-4. Specification Statements: Virtual Connect (cont.)**

Statement	Description
<a href="#">Virtual Connect Report Maximum</a>	Specifies the maximum number of virtual connect messages issued when virtual connect reporting is enabled.
<a href="#">Virtual Connect Semicolon As Colon</a>	Specifies identical net names containing a semicolon (;) to be virtually connected.

## Device Recognition Statement

Devices are defined for recognition and LVS comparison by the [Device](#) statement.

## Fracturing Statements

These statements pertain to database fracturing, which occurs prior to mask fabrication.

**Table 3-5. Fracturing Statements**

Statement	Description
<a href="#">Fracture HITACHI</a>	Converts layout input data into Hitachi form and outputs to a file for use in third-party mask preparation tools. Requires a Calibre FRACTUREh product license.
<a href="#">Fracture JEOL</a>	Converts layout input data into JEOL form and outputs to a file for use in third-party mask preparation tools. Requires a Calibre FRACTUREj product license.
<a href="#">Fracture MEBES</a>	Converts layout input data into MEBES form and outputs to a file for use in third-party mask preparation tools. Requires a Calibre FRACTUREm product license.
<a href="#">Fracture MICRONIC</a>	Converts layout input data into MICRONIC form and outputs to a file for use in third-party mask preparation tools. Requires a Calibre FRACTUREc product license.
<a href="#">Fracture NUFLARE</a>	Converts layout input data into VSB11/VSB12 form and outputs to a file for use in third-party mask preparation tools. Requires a Calibre FRACTUREt product license.
<a href="#">Fracture VBOASIS</a>	Converts layout input data into VB:OASIS form and outputs to a file for use in third-party mask preparation tools. Requires a Calibre FRACTUREv product license.
<a href="#">Fracture OASIS_MASK</a>	Converts layout input data into OASIS_MASK form (conforming to the SEMI P44-0708 standard) and outputs to a file for use in third-party mask preparation tools. As OASIS_MASK is a restriction of the OASIS format and a superset of the VB:OASIS format, it requires a Calibre FRACTUREv product license.

**Table 3-5. Fracturing Statements (cont.)**

Statement	Description
Litho File	Used to specify an inline file.
MDP Checkmap	Outputs single or multiple layers and datatypes with bin injection for large cells and is used primarily for OASIS bin injection
MDP Embed	Provides section-based processing for some SVRF commands without requiring a FRACTURE operation.
MDP Mapsize	Reads a job deck and merges individual chips and chip placements into a single chip and placement based on certain criteria.
MDP Maskopt	Modifies the input geometry to improve fracture quality. Constrained jog movements are applied to form polygons which, when fractured, will have fewer shot counts and fewer small figures than the initial fractured polygons.
MDP Oasis_Extent	Computes the extent of an OASIS layout.
MDPmerge	Optimizes the throughput of the mask writing machine, the parameter control like registration and CD, and enables the proximity correction for adjacent placements of individual chips based on various criteria
MDPstat	Analyzes Hitachi, JEOL and VSB11/VSB12 formatted data allowing you to assess the quality of the fracture output
MDPverify	Verifies fractured data by performing Boolean XOR comparison between databases of various types. Requires a Calibre MDPverify product license.

## Post-Design Correction Statements

These statements are used for optical and process correction.

**Table 3-6. Post-Design Correction Statements**

Operation or Statement	Description
Litho DenseOPC	Specifies to run Calibre nmOPC and to place the results on a layer you specify.
Litho File	Used to specify an inline file.
Litho OPC	Specifies to run Calibre OPCpro and to place the results on a layer you specify.
Litho OPCverify	Specifies to run Calibre OPCverify and to place the results on a layer you specify.

**Table 3-6. Post-Design Correction Statements**

Operation or Statement	Description
Litho ORC	Specifies to run Calibre ORC and to place the results on a layer you specify.
Litho Printimage	Specifies to run Calibre PRINTImage and to place the results on a layer you specify.
Litho PSMgate	Specifies to run Calibre PSMgate and to place the results on a layer you specify.
Opcbias	Calibre RET layer operation that performs variable isolated/dense line and via biasing. It oversizes or undersizes objects by moving individual edges pre-defined amounts. Requires a Calibre OPCpro or Calibre TDopc product license.
Opclineend	Calibre RET layer operation that performs variable line-end extension, from simple end extension to hammerhead creation. Requires a Calibre OPCpro or Calibre TDopc license.
Opcsbars	Calibre RET layer operation that creates scattering bars for optical correction. Requires a Calibre OPCbars license.

## Specification Statements

Specification statements describe the working environment for applications using the rule file. They control how data are processed.

The tables in the following sections list the specification statements used in Calibre and ICverify applications. For detailed descriptions of these statements, refer to the applicable “Reference Dictionary” subsection.

The Virtual Connect specification statements appear separately in Table 3-4.

## DFM Specification Statements

Table lists the DFM specification statements. This is a category of specification statements that applies to Calibre DFM applications (Calibre YieldAnalyzer and Calibre YieldEnhancer).

**Table 3-7. Specification Statements: DFM**

Statement	Description
DFM Assert	Asserts requirements on various SVRF objects and generates a compiler error if the requirements are not met.
DFM Database	Specifies the path to the DFM database directory to be created. Used only with calibre -dfm.

**Table 3-7. Specification Statements: DFM (cont.)**

Statement	Description
DFM Defaults	Allows default options to be specified that are then applied to subsequent SVRF statements.
DFM Function	Specifies a user-defined function that can be referenced within expressions for <a href="#">DFM Analyze</a> , <a href="#">DFM Property</a> , or other DFM Functions.
DFM Select Check	Specifies the rule checks to perform and write to the DFM database. Used only with calibre -dfm.
DFM Spec Fill	Creates a fill specification that can be used with <a href="#">DFM Fill</a> to generate fill layers.
DFM Spec Fill Optimizer	Defines a set of optimization criteria to be used in finding the optimal fill.
DFM Spec Fill Shape	Defines a fill shape and assigns it a name by which it can be referenced from within a DFM Spec Fill specification.
DFM Spec Spatial Sample	Defines a specification used to provide a rapid estimation of full-chip critical areas.
DFM Spec Via Redundancy	Defines a via redundancy analysis specification that can be referenced in a <a href="#">DFM Redundant Vias</a> operation.

## nmDRC Specification Statements

Table 3-8 lists the nmDRC specification statements. This a category of specification statements used for setting or modifying operating parameters for the nmDRC system. Used by all nmDRC applications except where noted.

**Table 3-8. Specification Statements: nmDRC**

Statement	Description
DRC Boolean Nosnap45	Specifies a merged shape must remain octagonal at the expense of inserting a 1 dbu edge.
DRC Cell Name	Specifies whether to append the cell name associated with each nmDRC result to its signature line in an ASCII nmDRC results database. Calibre nmDRC-H only.
DRC Cell Text	Specifies whether connectivity extraction text is read throughout the hierarchy of the input layout database. Calibre nmDRC-H only.
DRC Check Map	Specifies how Calibre nmDRC/nmDRC-H outputs the results for a specified rule check. Used for producing GDSII, OASIS, ASCII, or BINARY results databases.

**Table 3-8. Specification Statements: nmDRC (cont.)**

<b>Statement</b>	<b>Description</b>
DRC Check Text	Specifies the nmDRC rule check text mapping mode for nmDRC execution. In ICVerify, sets the initial nmDRC results database text mapping mode default for the \$check_drc() function.
DRC Exclude False Notch	Minimizes the possibility of false errors reported on notches during Calibre nmDRC-H checking for one-layer External operations. Use of the YES option is strongly discouraged in a production rule file.
DRC Incremental Connect	Enables incremental connectivity extraction for antenna checking and related applications.
DRC Incremental Connect Warning	Controls the generation of connectivity extraction warnings for [Not] Net operations that appear in a DRC Incremental Connect YES flow.
DRC Keep Empty	Specifies whether empty rule checks are to be retained in nmDRC execution. In ICVerify, sets the initial default to control writing rule checks that contain zero results to the nmDRC results database for the \$check_drc() function.
DRC Magnify Density	Specifies the default magnification factor for Density RDB databases.
DRC Magnify NAR	Specifies the default magnification factor for Net Area Ratio RDB databases.
DRC Magnify Results	Specifies that the nmDRC results database is magnified by the given value as it is being output by Calibre nmDRC.
DRC Map Text	Specifies to transfer text objects in the input layout database to the nmDRC results database. Used only with Calibre nmDRC-H.
DRC Map Text Depth	Specifies the hierarchical depth from which to read text objects for the DRC Map Text YES specification statement.
DRC Map Text Layer	Specifies layers from which to map text objects to the nmDRC-H results database.
DRC Maximum Cell Name Length	Specifies the maximum length of cell (and placement) names in GDSII and OASIS DRC results databases. Used only with Calibre nmDRC-H.
DRC Maximum Results	Specifies the nmDRC rule check maximum result count for nmDRC execution. In ICVerify, sets the initial default maximum number of results that the \$check_drc() function can place in the nmDRC results database for each rule check statement.

**Table 3-8. Specification Statements: nmDRC (cont.)**

<b>Statement</b>	<b>Description</b>
DRC Maximum Unattached Label Warnings	Specifies the nmDRC maximum number of unattached label warnings issued by the connectivity extraction module.
DRC Maximum Vertex	Specifies the maximum vertex count for polygon DRC results to be written to the nmDRC results database. Used only in Calibre nmDRC/nmDRC-H.
DRC Print Area	Computes the flat area for each specified layer and prints it to the transcript when the specified layers are generated. Used only in Calibre nmDRC/nmDRC-H.
DRC Print Perimeter	Computes the flat perimeter for each specified layer and prints it to the transcript when the specified layers are generated. Used only in Calibre nmDRC/nmDRC-H.
DRC Results Database	Specifies the filename and type of the nmDRC results database. Used only in Calibre nmDRC/nmDRC-H.
DRC Results Database Libname	Specifies the LIBNAME record of GDSII results databases.
DRC Results Database Precision	Specifies the precision value in the Calibre nmDRC results database. Has no effect on coordinate values output to the database.
DRC Select Check	Specifies rule checks or groups to be executed.
DRC Select Check By Layer	Specifies rule checks to be executed by layer names.
DRC Summary Report	Specifies the nmDRC summary report filename and writing method for nmDRC execution. In ICVerify, sets the initial summary report filename and file opening mode for the \$check_drc() function.
DRC Tolerance Factor	Specifies a tolerance factor for dimensional check operations (External, Internal, and Enclosure).
DRC Tolerance Factor NAR	Specifies the tolerance value for comparing Net Area Ratio computed values to the constraint values.
DRC Unselect Check	Specifies rule checks or groups to be excluded from the run.
DRC Unselect Check By Layer	Specifies rule checks to be excluded from a run, based upon layer names.
Group	Names a collection of rule check statements.

## ERC Specification Statements

Table 3-9 lists the Electrical Rule Check specification statements. These are specification statements used for setting or modifying operating parameters for the ERC system. ERC is performed as a part of LVS.

**Table 3-9. Specification Statements: ERC**

Statement	Description
ERC Cell Name	Specifies whether to append the cell name associated with each ERC result to its signature line in an ASCII ERC results database. Used only for Calibre nmLVS-H.
ERC Check Text	Specifies the amount and type of text to appear in ERC results databases.
ERC Keep Empty	Specifies whether empty rule checks (rule checks with zero results) are retained in ERC execution.
ERC Maximum Results	Specifies the maximum number of results for an individual rule check in ERC execution.
ERC Maximum Vertex	Specifies the maximum vertex count of any ERC result polygon Calibre writes to the ERC results database.
ERC Path Also	Allows capacitors, diodes, and bipolar transistors to be considered part of a “path” in ERC Pathchk.
ERC Pathchk	Reports nets in the layout that do or do not have a path to power, ground or labeled nets, or that satisfy a combination of these conditions.
ERC Results Database	Specifies the pathname and type of ERC results database Calibre creates.
ERC Select Check	Selects rule check statements and rule check groups to be executed by the ERC process.
ERC Summary Report	Specifies the report filename and writing method for ERC execution.
ERC Unselect Check	Selects rule check statements and rule check groups not to be executed by the ERC process.
Group	Names a collection of rule check statements.
LVS Execute ERC	Specifies whether or not ERC is performed by Calibre nmLVS.

There is one ERC layer operation called [Pathchk](#). Its use is preferred over the ERC Pathchk statement.

## Layout Database Specification Statements

Table 3-10 lists the layout database specification statements. These are specification statements used to describe contents of layout databases, pathnames, manipulation of layout objects, global flagging of error conditions in the layout, hierarchy management of cells and layers, and layout input control.

These statements are used in all Calibre and ICVerify applications except where noted.

**Table 3-10. Specification Statements: Layout Database**

Statement	Description
<a href="#">Exclude Acute</a>	Specifies layers to be excluded from Flag Acute and Drawn Acute checking. Not used in ICVerify.
<a href="#">Exclude Angled</a>	Specifies layers to be excluded from Flag Angled and Drawn Angled checking. Not used in ICVerify.
<a href="#">Exclude Cell</a>	Specifies cell names to exclude from the layout database, that is, layout cells that are not to be processed.
<a href="#">Exclude Offgrid</a>	Specifies layers to be excluded from Flag Offgrid and Drawn Offgrid checking. Not used in ICVerify.
<a href="#">Exclude Skew</a>	Specifies layers to be excluded from Flag Skew and Drawn Skew checking. Not used in ICVerify.
<a href="#">Expand Cell</a>	Specifies the cells having instances to be flattened one level into the space of the cells in which the specified cells are placed. Used in hierarchical Calibre applications.
<a href="#">Expand Cell Text</a>	Specifies that the text from placements of the specified source cell be used higher up in the layout hierarchy. Used in hierarchical Calibre applications.
<a href="#">Flag Acute</a>	Controls acute angle flagging for original layout database shapes read by the verification application.
<a href="#">Flag Angled</a>	Controls flagging of edges that are not orthogonal to the database axes for original layout database shapes read by the verification application. Not used in ICVerify.
<a href="#">Flag Nonsimple [Polygon]</a>	Controls non-simple polygon flagging when original layout database shapes are input. Not used in ICVerify.
<a href="#">Flag Nonsimple Path</a>	Controls non-simple path flagging when original layout database shapes are input. Not used in ICVerify.
<a href="#">Flag Offgrid</a>	Controls off-grid vertex flagging for original layout database shapes read by the verification application.
<a href="#">Flag Skew</a>	Controls skew edge flagging for original layout database shapes read by the verification application. Skew edges do not include 45-degree edges.

**Table 3-10. Specification Statements: Layout Database (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">Flatten Cell</a>	Specifies the cells having instances to be flattened into the space of the cells in which the specified cells are placed. Used only in hierarchical Calibre applications.
<a href="#">Flatten Inside Cell</a>	Specifies that the internal hierarchy of each cell in the parameter list is to be flattened during the hierarchical database construction phase. Used only in hierarchical Calibre applications.
<a href="#">Layer</a>	Defines the name of an original layer or an original layer set in terms of layer numbers or other layer names in the rule file.
<a href="#">Layer Directory</a>	Specifies a directory for disk-based layers. Not used in nmLVS or Calibre xRC applications.
<a href="#">Layer Map</a>	Builds datatype or texttype maps from GDSII or OASIS input for Calibre. Not used in ICverify.
<a href="#">Layer Resolution</a>	Overrides the default Resolution specification statement parameters for any number of original layers during off-grid vertex checking for Flag Offgrid, Drawn Offgrid, or the ICrules CHEck DRc -FLAGOFFGRID option.
<a href="#">Layout Allow Duplicate Cell[s]</a>	Specifies whether multiple records for the same layout cell are allowed for the input layout database. Not used in ICverify.
<a href="#">Layout Base Cell</a>	Specifies base cell names for gate array designs. Used only in hierarchical Calibre applications.
<a href="#">Layout Base Layer</a>	Specifies that cell instances that contain layers not specified in this statement are ignored by the system when distinguishing the design style. Used only in hierarchical Calibre applications. Counterpart to Layout Top Layer.
<a href="#">Layout Bump2</a>	Specifies that the layer numbers in a database are increased by a specified value. For use with dual database capability. Not used in ICverify.
<a href="#">Layout Case</a>	Specifies whether the layout should be processed in a case-sensitive manner. Not used in nmDRC applications.
<a href="#">Layout Cell List</a>	Specifies a cell list for specification statements that use cell lists. Used only in hierarchical Calibre applications.
<a href="#">Layout Clone By Layer</a>	Specifies to clone cells during the hierarchical database (HDB) construction phase if they are marked.
<a href="#">Layout Clone Rotated Placements</a>	Specifies whether to clone cells that are rotated by 90 or 270 degrees during HDB construction.
<a href="#">Layout Clone Transformed Placements</a>	Specifies whether to use cell cloning for all rotated or transformed cells during HDB construction.

**Table 3-10. Specification Statements: Layout Database (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">Layout Depth</a>	Specifies the hierarchical depth for fetching shapes from the layout database for Calibre. Not used in ICVerify.
<a href="#">Layout Error On Input</a>	Specifies whether certain warnings reported when reading the layout database are to become fatal errors. Not used in ICVerify.
<a href="#">Layout Ignore Text</a>	Specifies to discard layout database text for cells in a given list. Used only in hierarchical Calibre applications.
<a href="#">Layout Input Exception RDB</a>	Specifies a results database for storing exception objects from specified Layout Input Exception Severity statements.
<a href="#">Layout Input Exception Severity</a>	Specifies how layout input exceptions are handled (for instance, fatal errors and warnings). Not used in ICVerify.
<a href="#">Layout Magnify</a>	Specifies that the input layout database is magnified by the given value as it is being read into the Calibre application.
<a href="#">Layout Merge On Input</a>	Directs the layout reader to merge polygons on a per-cell, per-layer basis as it reads the stream into memory. Not used in ICVerify.
<a href="#">Layout Path</a>	Specifies the layout database pathname(s) for the database type specified by the Layout System statement. Not used in ICVerify
<a href="#">Layout Path2</a>	Specifies the layout database pathname(s) for the database type specified by the Layout System2 statement. For use with dual database capability. Not used in ICVerify.
<a href="#">Layout Place Cell</a>	Specifies to construct a GDSII or OASIS database from parameters given in this statement.
<a href="#">Layout Polygon</a>	Allows original database shapes to be specified directly in the rule file. Used only in Calibre.
<a href="#">Layout Precision</a>	Allows specification of an alternate precision value for checking the precision of input layout databases in Calibre.
<a href="#">Layout Preserve Cell List</a>	Specifies to protect certain layout cells from most hierarchy modifications, including automatic expansion, during connectivity extraction in Calibre nmLVS-H.
<a href="#">Layout Primary</a>	Specifies the layout database topcell name for Calibre. Not used in ICVerify.
<a href="#">Layout Primary2</a>	Specifies the layout database topcell name for Calibre. For use with dual database capability. Not used in ICVerify.
<a href="#">Layout Process Box Record</a>	Specifies whether BOX and BOXTYPE records in GDSII input layout databases are processed. Not used in ICVerify.
<a href="#">Layout Process Node Record</a>	Specifies whether NODE and NODETYPE records in GDSII input layout databases are processed. Not used in ICVerify.

**Table 3-10. Specification Statements: Layout Database (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">Layout Property Audit</a>	Specifies property names and values to be inserted, appended, or modified in an OASIS nmDRC results database.
<a href="#">Layout Property Text</a>	Instructs Calibre to treat properties attached to shapes as text objects with PROPVALUE as the name of the text object.
<a href="#">Layout Property Text OASIS</a>	Instructs Calibre to treat OASIS properties as input layout database text.
<a href="#">Layout Rename Cell</a>	Specifies cells to be renamed in the geometric input layout database. Not used in ICverify.
<a href="#">Layout Rename ICV</a>	Specifies that Calibre use unique identification of ICV cells across multiple files.
<a href="#">Layout Rename Text</a>	Specifies text values to be edited or replaced. The statement edits the text value as it is being read. Not used in ICverify.
<a href="#">Layout System</a>	Specifies the layout database format. Not used in ICverify.
<a href="#">Layout System2</a>	Specifies the layout database format. For use with dual database capability. Not used in ICverify.
<a href="#">Layout Text</a>	Allows a layout database text object to be specified directly in the rule file. Not used in ICverify.
<a href="#">Layout Text File</a>	Allows a layout database text object to be specified in a file. This is used when there are many text objects to specify. Not used in ICverify.
<a href="#">Layout Top Layer</a>	Specifies that cell instances which contain only the specified layers are ignored by the system when distinguishing the design style. Used in hierarchical Calibre applications. Counterpart to Layout Base Layer.
<a href="#">Layout Use Database Precision</a>	Specifies to use the input database precision rather than the rule file precision.
<a href="#">Layout Windel</a>	Specifies a single polygon window that defines the <i>exclusion</i> of input shapes and text from use in rule checks. Not used in ICverify.
<a href="#">Layout Windel Cell</a>	Specifies multiple windows using a list of cell names in order to <i>exclude</i> input shapes and text from use in rule checks. Not used in ICverify.
<a href="#">Layout Windel Layer</a>	Specifies multiple windows from a list of layers in order to <i>exclude</i> input shapes and text from use in rule checks. Not used in ICverify.

**Table 3-10. Specification Statements: Layout Database (cont.)**

Statement	Description
Layout Window	Specifies a single polygon window that defines the <i>inclusion</i> of input shapes and text for use in rule checks. Not used in ICVerify.
Layout Window Cell	Specifies multiple windows using a list of cell names in order to <i>include</i> input shapes and text for use in rule checks. Not used in ICVerify.
Layout Window Clip	Specifies whether area-based filtering in Calibre will be Boolean or topological. Not used in ICVerify.
Layout Window Layer	Specifies multiple windows from a list of layers in order to <i>include</i> input shapes and text for use in rule checks. Not used in ICVerify.
Polygon	Allows original database shapes to be specified directly in the rule file.
Port Depth	Specifies the hierarchical depth for reading of port objects for use in the top-level cell. Not used in ICVerify or by nmDRC applications.
Port Layer Polygon	For input layout databases, causes shapes on the specified layer(s) to be read and treated as geometric ports. Not used in ICVerify or by nmDRC applications.
Port Layer Text	For input layout databases, causes text objects on the specified layer(s) to be read and treated as text ports. Not used in ICVerify or by nmDRC applications.
Precision	Defines the ratio of database units to user-defined units.
Push Cell	Specifies cell names whose placements are to be completely flattened, then pushed to the lowest possible level by hierarchical Calibre applications.
Resolution	Defines the layout grid step-size (the number of database units on which any original geometry is required to be aligned) in either the x-direction or the y-direction.
Snap Offgrid	Snaps original layer shapes to the grid specified in the Resolution specification statement or the appropriate Layer Resolution specification statement, if present.
Text	Allows layout database text objects (free-standing text) to be specified directly in the rule file. It also allows text objects read from the layout database to be edited.
Text Depth	Specifies the hierarchical depth for fetching text objects from the layout database for the connectivity extractor.

**Table 3-10. Specification Statements: Layout Database (cont.)**

Statement	Description
Text Layer	Specifies the layers in the database from which free-floating text is read for connectivity extraction only.

## LVS Specification Statements

Table 3-11 lists the LVS specification statements. These are specification statements used in layout versus schematic checking.

**Table 3-11. Specification Statements: LVS**

Statement	Description
Device	Defines devices to be recognized and used in LVS comparison. Not a specification statement per-se, but used by LVS processes.
Hcell	Specifies a pair of hcells—one from layout and one from source—that correspond.
Layout Preserve Case	Specifies whether matching net names that differ only by case are treated as identical or different during connectivity extraction.
LVS Abort On ERC Error	Specifies whether nmLVS terminates when ABORT LVS keywords are specified in ERC Select Check statements. Used only in Calibre nmLVS/nmLVS-H applications.
LVS Abort On Softchk	Specifies whether to abort processing if a Softchk violation is found.
LVS Abort On Supply Error	Specifies whether to abort processing when connectivity extraction detects a problem with the power or ground nets. Not used in ICverify or Calibre nmLVS-H.
LVS All Capacitor Pins Swappable	Specifies whether all capacitor pins should be considered swappable. In ICverify, sets the value of application variable lvs_all_capacitor_pins_swappable.
LVS Annotate Devices	Specifies to annotate devices in the extracted netlist with properties stored on an annotation layer.
LVS Auto Expand Hcells	Specifies whether to expand hcells automatically during connectivity extraction in order to optimize performance.
LVS Box	Specifies cells with contents to be treated as black boxes.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Builtin Device Pin Swap</a>	Specifies whether nmLVS should apply default pin swappability rules in built-in devices.
<a href="#">LVS Builtin MOS NRD_NRS</a>	Specifies whether NRS/NRD properties are treated as built-in during parallel MOS reduction.
<a href="#">LVS Cell List</a>	Specifies a list of cells for overrides of LVS rules. Used only in Calibre nmLVS-H.
<a href="#">LVS Cell Supply</a>	Controls propagation of voltage signals from lower level ports.
<a href="#">LVS Center Device Pins</a>	Identifies and returns centered pin location coordinates.
<a href="#">LVS Check Port Names</a>	Specifies whether to check the names of matched ports.
<a href="#">LVS Compare Case</a>	Controls case sensitivity in the LVS circuit comparison stage.
<a href="#">LVS Component Subtype Property</a>	Specifies the property name to be used in determining component subtypes for source instances. In ICverify, sets the value of application variable lvs_component_subtype_property.
<a href="#">LVS Component Type Property</a>	Specifies property names to be used in determining component types for source instances. In ICverify, sets the value of application variable lvs_component_type_property.
<a href="#">LVS Cpoint</a>	Specifies a correspondence point between a layout net and a source net.
<a href="#">LVS Device Type</a>	Specifies instances having non-built-in device names to be treated as instances of built-in devices.
<a href="#">LVS Discard Pins By Device</a>	Specifies whether nmLVS should examine rule file Device operations in order to discard extra pins from input devices during circuit comparison.
<a href="#">LVS Downcase Device</a>	Specifies that nmLVS use lowercase representations of Device element names, model names, and pin names.
<a href="#">LVS EDDM Process M</a>	Specifies that ICtrace apply M properties to other properties in an EDDM database.
<a href="#">LVS Exact Subtypes</a>	Specifies that circuit comparison use both device type and subtype in the initial match.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Exclude Hcell</a>	Specifies that Calibre nmLVS-H ignores the specified hcells in all circumstances. These hcells are also ignored by the Query Server.
<a href="#">LVS Execute ERC</a>	Specifies whether Calibre performs ERC operations during the circuit extraction phase.
<a href="#">LVS Expand Seed Promotions</a>	Instructs the Calibre nmLVS-H circuit comparison module to expand hcells that were subject to seed promotion during hierarchical device recognition.
<a href="#">LVS Expand Unbalanced Cells</a>	Specifies whether Calibre nmLVS-H expands hcells that are instantiated a different number of times in the source and layout.
<a href="#">LVS Filter</a>	Filters out source and layout instances having properties with the specified values.
<a href="#">LVS Filter Unused Bipolar</a>	Specifies whether to filter unused bipolar transistors. In ICverify, sets the value of application variable lvs_filter_unused_bipolar_transistors.
<a href="#">LVS Filter Unused Capacitors</a>	Specifies whether to filter unused capacitors. In ICverify, sets the value of application variable lvs_filter_unused_capacitors.
<a href="#">LVS Filter Unused Diodes</a>	Specifies whether to filter unused diodes. In ICverify, sets the value of application variable lvs_filter_unused_diodes.
<a href="#">LVS Filter Unused MOS</a>	Specifies whether to filter unused MOS transistors. In ICverify, sets the value of application variable lvs_filter_unused_mos_transistors.
<a href="#">LVS Filter Unused Option</a>	Controls the filtering process of unused devices.
<a href="#">LVS Filter Unused Resistors</a>	Specifies whether to filter unused resistors. In ICverify, sets the value of application variable lvs_filter_unused_resitors.
<a href="#">LVS Flatten Inside Cell</a>	Specifies whether to flatten the hierarchy of cells during LVS-H comparison or in hierarchical PERC applications.
<a href="#">LVS Global Layout Name</a>	Specifies to report inconsistent connections that involve global layout signals. Used only in Calibre nmLVS-H.
<a href="#">LVS Globals Are Ports</a>	Specifies whether nmLVS treats global nets as ports of the top level cell.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Ground Name</a>	Specifies a list of zero or more ground net names. In ICVerify, sets the value of application variable lvs_ground_names.
<a href="#">LVS Heap Directory</a>	Specifies the directory where temporary heap files are to be stored and its capacity.
<a href="#">LVS Ignore Ports</a>	Specifies whether the LVS comparison algorithm should ignore ports. In ICVerify, sets the value of application variable lvs_ignore_ports.
<a href="#">LVS Ignore Trivial Named Ports</a>	Controls the handling of ports to lower-level cells that are not connected to any devices, and that have corresponding names elsewhere in the design.
<a href="#">LVS Inject Logic</a>	Specifies whether to perform logic injection during circuit comparison. Used only in Calibre nmLVS-H.
<a href="#">LVS Isolate Shorts</a>	Specifies whether to enable short isolation.
<a href="#">LVS Map Device</a>	Specifies to map original device component subtypes to a new component subtype. Used only in Calibre.
<a href="#">LVS MOS Swappable Properties</a>	Specifies swappable (user-defined) source and drain pin properties for Device pushdown when using LVS Push Devices SEPARATE PROPERTIES. Used only in Calibre nmLVS-H.
<a href="#">LVS Netlist All Texted Pins</a>	Specifies whether to netlist internally-floating, texted pins. Used only in Calibre nmLVS-H.
<a href="#">LVS Netlist Allow Inconsistent Model</a>	Specifies to allow certain MOS devices to use model names that are inconsistent with Device statement conventions. Use of this statement is discouraged.
<a href="#">LVS Netlist Box Contents</a>	Specifies whether to write internal contents of layout LVS Box cells to the output netlist.
<a href="#">LVS Netlist Comment Coded Properties</a>	Specifies whether the Calibre nmLVS-H circuit netlister should use the comment-coded parameters \$A and \$P (for capacitors) or \$W and \$L (for bipolar transistors).
<a href="#">LVS Netlist Comment Coded Substrate</a>	Specifies whether the Calibre nmLVS-H circuit netlister should use the comment-coded parameter \$SUB to represent substrate pins in 3 pin diodes, 3 pin capacitors, 3 pin resistors, and 4 pin bipolar devices.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Netlist Unnamed Box Pins</a>	Specifies whether the Calibre nmLVS-H circuit netlister includes certain unnamed pins of LVS Box cells in the output netlist.
<a href="#">LVS NL Pin Locations</a>	Controls the flat layout netlist created by the -nl command line option of Calibre nmLVS. It instructs flat LVS to output pin locations and related information for user defined devices in the -nl netlist.
<a href="#">LVS Non User Name</a>	Specifies instance, net, or port names (except power or ground names) that should not be treated as user-given names in LVS, even if they otherwise fit the criteria for user-given names.
<a href="#">LVS Out Of Range Exclude Zero</a>	Specifies how Trace Property ... ABSOLUTE tolerance warnings are handled when property values are zero.
<a href="#">LVS Pin Name Property</a>	Specifies for ICtrace a prioritized list of zero or more property names to be used in determining pin names for source instance pins. Sets the value of application variable lvs_pin_name_property.
<a href="#">LVS Power Name</a>	Specifies a list of zero or more power net names. In ICverify, sets the value of application variable lvs_power_names.
<a href="#">LVS Precise Interaction</a>	Specifies whether to compute precise interaction regions during hierarchical device recognition. Used only in Calibre nmLVS-H.
<a href="#">LVS Preserve Box Cells</a>	Specifies how LVS Box cell contents are written to an extracted layout netlist. Used only in Calibre nmLVS-H.
<a href="#">LVS Preserve Box Ports</a>	Specifies how LVS Box cell ports are handled during high-short resolution. Used only in Calibre nmLVS-H.
<a href="#">LVS Preserve Floating Top Nets</a>	Controls preservation of floating named top-level nets in the extracted layout SPICE netlist.
<a href="#">LVS Preserve Parameterized Cells</a>	Prevents flattening of parameterized subcircuits in Calibre nmLVS-H.
<a href="#">LVS Property Initialize</a>	Instructs nmLVS to add numeric properties simultaneously to instances in the layout and source input databases, and initialize those properties to known values.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Property Map</a>	Instructs nmLVS to re-map property names in the specified input database.
<a href="#">LVS Property Resolution Maximum</a>	Specifies that nmLVS use property values to resolve all groups of ambiguous elements that contain no more than a maximum number of elements.
<a href="#">LVS Push Devices</a>	Specifies whether to push device seed polygons down the hierarchy after seed promotion. Used only in Calibre nmLVS-H.
<a href="#">LVS Recognize Gates</a>	Specifies whether to recognize logic gates from transistor-level data. In ICVerify, sets the values of application variables lvs_recognize_gates or lvs_recognize_only_simple_gates.
<a href="#">LVS Recognize Gates Tolerance</a>	Specifies whether to limit gate recognition using device property tolerance checking.
<a href="#">LVS Reduce</a>	Provides generic device reduction instructions to nmLVS.
<a href="#">LVS Reduce Parallel Bipolar</a>	Specifies whether to reduce bipolar transistors connected in parallel into single transistors. In ICVerify, sets the value of application variable lvs_reduce_parallel_bipolar.
<a href="#">LVS Reduce Parallel Capacitors</a>	Specifies whether to reduce capacitor devices connected in parallel into single capacitors. In ICVerify, sets the value of application variable lvs_reduce_parallel_capacitors.
<a href="#">LVS Reduce Parallel Diodes</a>	Specifies whether to reduce diodes connected in parallel into single diodes. In ICVerify, sets the value of application variable lvs_reduce_parallel_diodes.
<a href="#">LVS Reduce Parallel MOS</a>	Specifies whether to reduce MOS transistors connected in parallel into single transistors. In ICVerify, sets the value of application variable lvs_reduce_parallel_mos.
<a href="#">LVS Reduce Parallel Resistors</a>	Specifies whether to reduce resistor devices connected in parallel into single resistors. In ICVerify, sets the value of application variable lvs_reduce_parallel_resistors.
<a href="#">LVS Reduce Semi Series MOS</a>	Specifies whether to reduce MOS devices connected in series including those with bypass nets. In ICVerify, sets the value of application variable lvs_reduce_semi_series_mos.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Reduce Series Capacitors</a>	Specifies whether to reduce capacitor devices connected in series into single capacitors. In ICverify, sets the value of application variable lvs_reduce_series_capacitors.
<a href="#">LVS Reduce Series MOS</a>	Specifies whether to reduce MOS devices connected in series. In ICverify, sets the value of application variable lvs_reduce_series_mos.
<a href="#">LVS Reduce Series Resistors</a>	Specifies whether to reduce resistor devices connected in series into single resistors. In ICverify, sets the value of application variable lvs_reduce_series_resistors.
<a href="#">LVS Reduce Split Gates</a>	Specifies whether to reduce MOS split gates formed as serial-up or serial-down structures into single gate structures. In ICverify, sets the value of application variable lvs_reduce_split_gates.
<a href="#">LVS Reduction Priority</a>	Controls which type of device reduction to perform when both series and parallel reduction are possible on the same group of devices.
<a href="#">LVS Report</a>	Specifies the report pathname. In ICverify, sets the value of application variable lvs_default_report_name.
<a href="#">LVS Report Maximum</a>	Specifies an upper limit on the number of items that nmLVS prints out in various sections of its report. In ICverify, sets the value of application variable lvs_report_list_limit.
<a href="#">LVS Report Option</a>	Controls the level of detail and verbosity of a nmLVS report, as well as some other aspects of the report.
<a href="#">LVS Report Units</a>	Specifies whether units should be reported with built-in device property values.
<a href="#">LVS Report Warnings Hcell Only</a>	Specifies whether connectivity extraction warnings should be reported for cells not specified in an hcell file or the Hcell statement. Used only in Calibre nmLVS-H.
<a href="#">LVS Report Warnings Top Only</a>	Specifies whether connectivity extraction warnings should be reported only for the top-level cell. Used only in Calibre nmLVS-H.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Reverse WL</a>	Specifies to reverse the default L,W order in MOS devices in SPICE netlists and within SPICE-like properties in EDDM and Direct Mode ICVerify designs.
<a href="#">LVS Show Seed Promotions</a>	Specifies whether instances in the design where devices are formed by interaction of polygons from different levels of hierarchy are found. Used only by Calibre nmLVS-H.
<a href="#">LVS Show Seed Promotions Maximum</a>	Specifies the maximum number of polygons that will appear in any check performed by the LVS Show Seed Promotions statement. Used only by Calibre nmLVS-H.
<a href="#">LVS Signature Maximum</a>	Specifies that the LVS signature algorithm compute signatures for neighborhoods no bigger than a maximum size.
<a href="#">LVS Soft Substrate Pins</a>	Specifies whether substrate and bulk pins have a standard effect on how circuit elements are matched.
<a href="#">LVS Softchk</a>	Selects polygons involved in conflicting connections from all Sconnect operations.
<a href="#">LVS Spice Allow Duplicate Subcircuit Names</a>	Specifies whether the LVS SPICE parser will allow duplicate subcircuit definitions with a warning, or disallow them.
<a href="#">LVS Spice Allow Floating Pins</a>	Specifies whether to allow floating pins in subcircuit calls in SPICE netlists.
<a href="#">LVS Spice Allow Inline Parameters</a>	Specifies whether to allow inline parameter references for subcircuit calls (X calls) in SPICE netlists. Used only in Calibre nmLVS/nmLVS-H.
<a href="#">LVS Spice Allow Unquoted Strings</a>	Specifies whether the LVS SPICE parser should allow unquoted string values in parameter assignments.
<a href="#">LVS Spice Conditional LDD</a>	Controls how LVS SPICE reader processes \$LDD designators in M elements within SPICE netlists.
<a href="#">LVS Spice Cull Primitive Subcircuits</a>	Specifies whether to ignore primitive subcircuits having names that do not appear in the rule file during LVS comparison.
<a href="#">LVS Spice Implied MOS Area</a>	Specifies whether to calculate implied device areas for “M” elements in SPICE netlists.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Spice Multiplier Name</a>	Specifies parameter names that serve as multiplier factors in SPICE netlists instead of the standard SPICE parameter name M.
<a href="#">LVS Spice Option</a>	Specifies the behavior of the LVS SPICE parser.
<a href="#">LVS Spice Override Globals</a>	Specifies whether nmLVS allows pin assignments to override global signals in SPICE netlists.
<a href="#">LVS Spice Prefer Pins</a>	Specifies whether to prefer subcircuit pins over global nets when resolving net names in SPICE netlists.
<a href="#">LVS Spice Redefine Param</a>	Specifies whether .PARAM values may be redefined in a SPICE netlist.
<a href="#">LVS Spice Rename Parameter</a>	Specifies that SPICE parameters are renamed from an existing name to a different name during nmLVS comparison in all places where the existing name appears in the source and layout netlists. Used only in Calibre nmLVS/nmLVS-H.
<a href="#">LVS Spice Replicate Devices</a>	Specifies whether the LVS SPICE parser should physically replicate device elements in a SPICE netlist that own the M parameter.
<a href="#">LVS Spice Scale X Parameters</a>	Specifies whether LVS comparison should apply .OPTIONS SCALE factors to built-in devices instantiated as X elements in SPICE netlists.
<a href="#">LVS Spice Slash Is Space</a>	Specifies whether the slash character should be treated as white space within subcircuit definitions and subcircuit calls in SPICE netlists.
<a href="#">LVS Spice Strict WL</a>	Controls the parsing of width and length parameters for MOSFETs in SPICE netlists.
<a href="#">LVS Split Gate Ratio</a>	Specifies to check property ratios in split gate structures.
<a href="#">LVS Strict Subtypes</a>	Instructs nmLVS applications to control the processing of component subtypes in LVS, based on whether unspecified component subtypes are matched to instances with specified subtypes
<a href="#">LVS Summary Report</a>	Specifies that nmLVS-H generate a summary report file after a circuit comparison run.
<a href="#">LVS Write Injected Layout Netlist</a>	Specifies for nmLVS-H to output a SPICE netlist from the layout, including injected logic.

**Table 3-11. Specification Statements: LVS (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">LVS Write Injected Source Netlist</a>	Specifies for nmLVS-H to output a SPICE netlist from the schematic, including injected logic.
<a href="#">LVS Write Layout Netlist</a>	Specifies to output a transformed SPICE netlist from the layout.
<a href="#">LVS Write Source Netlist</a>	Specifies to output a transformed SPICE netlist from the schematic.
<a href="#">Mask Results Database</a>	Specifies the pathname for the mask results database and the type of information to be excluded from that database. In ICverify, sets the value of application variable mask_default_database_name.
<a href="#">Mask SVDB Directory</a>	Creates a Standard Verification Database Directory (SVDB) in Calibre.
<a href="#">Port Depth</a>	Specifies the hierarchical depth for reading of port objects for use in the top-level cell. Not used in ICverify.
<a href="#">Port Layer Polygon</a>	For input layout databases, causes shapes on the specified layer(s) in the top-level cell to be read and treated as geometric ports. Not used in ICverify.
<a href="#">Port Layer Text</a>	For input layout databases, causes text objects on the specified layer(s) in the top-level cell to be read and treated as text ports. Not used in ICverify.
<a href="#">Source Case</a>	Specifies whether the source should be processed in a case-sensitive manner.
<a href="#">Source Path</a>	Specifies the source database pathname for Calibre. Not used in ICverify.
<a href="#">Source Primary</a>	Specifies a design filename, subcircuit, or cell name for SPICE or CNET source system for Calibre. Not used in ICverify.
<a href="#">Source System</a>	Specifies the layout database type for Calibre. Not used in ICverify.
<a href="#">Trace Property</a>	Guides device parameter comparison. Specifies the rules that control which properties are traced and how the comparison is done.

## PERC Specification Statements

Table 3-12 lists the specification statements for Programmable Electrical Rule Checking (PERC).

**Table 3-12. Specification Statements: PERC**

Statement	Description
PERC Load	Specifies the PERC initialization procedures and rule checks to execute during a PERC run.
PERC Netlist	Specifies the input netlist for PERC runs.
PERC Pattern Path	Specifies a PERC pattern file.
PERC Property	Specifies properties made available for PERC, even if the properties are not otherwise needed for LVS.
PERC Report	Specifies the pathname of the PERC report file.
PERC Report Maximum	Specifies the number of reported items in the PERC report file.
PERC Report Option	Specifies the level of detail in the PERC Report.
PERC Report Placement List Maximum	Specifies the maximum number of elements in placement lists in the PERC Report.
PERC Waiver Path	Specifies to use waivers during the PERC run.

## PEX Specification Statements

Table 3-13 lists the PEX specification statements. These statements are used in parasitic extraction applications.

**Table 3-13. Specification Statements: PEX**

Statement	Description
Capacitance Order	Determines the vertical order of layers for capacitance calculations from bottom to top.
Parasitic Variation	Modifies nominal resistance calculations to more accurately model the fabrication process.
PEX 3DReference	Specifies the optional parameters to be used by the capacitance field solver.
PEX Alias	Specifies layers that use the capacitance, resistance, and in-die variation rules of the first layer.
PEX BA Mapfile	Specifies a file containing the device and pin name mappings used to backannotate the extracted parasitics to a pre-layout simulation netlist.

**Table 3-13. Specification Statements: PEX (cont.)**

<b>Statement</b>	<b>Description</b>
<b>PEX Bulk Layer</b>	Specifies the layer or layers used for connecting to the bulk pins of transistors.
<b>PEX Contact Capacitance</b>	Switches on or off the calculations for contact capacitance.
<b>PEX Corner</b>	Specifies the names of the foundry-supplied process corners and is generated by the calibration process using the xCalibrate rule file generator.
<b>PEX Corner Custom</b>	Optional statement that specifies variations on foundry-supplied process corners. The rules calculating parasitic capacitance and resistance must include functions for process variation.
<b>PEX CMP Mode</b>	Specifies the type of CMP mode used to handle layer thickness variation during parasitic extraction.
<b>PEX DEF Extract Cell Obstructions</b>	Specifies that obstruction geometries are treated as layout objects and includes them when calculating parasitic capacitance.
<b>PEX DEF Map</b>	Maps a DEF layer name to a layer name used in the SVRF rule file.
<b>PEX Density Estimate</b>	Specifies an estimated value of the local density of a layer in the layout.
<b>PEX Density Window</b>	Specifies the window size, in microns, for density computation.
<b>PEX Elayer</b>	Groups together layers sharing the same height profiles or derived layers.
<b>PEX Exclude Distributed</b>	Specifies to limit distributed extraction to the net names <i>not</i> listed.
<b>PEX Exclude Lumped</b>	Specifies to limit lumped extraction to the net names <i>not</i> listed. That is, specified nets are excluded from processing.
<b>PEX Extract Exclude</b>	Specifies to exclude listed net names from extraction results.
<b>PEX Extract Floating Nets</b>	Specifies how floating nets like metal fill are handled during extraction.
<b>PEX Extract Include</b>	Specifies to limit extraction to the listed net names.
<b>PEX Extract Rgate</b>	Specifies how to calculate the gate resistance of a transistor.

**Table 3-13. Specification Statements: PEX (cont.)**

Statement	Description
<a href="#">PEX Extract Temperature</a>	Specifies a temperature dependency for the extracted resistance.
<a href="#">PEX Fieldsolver Boundary</a>	Specifies the boundary conditions used by the capacitance solver.
<a href="#">PEX Fieldsolver Cell_array</a>	Specifies the boundary conditions used by the capacitance solver and grounds all the conductors in the image.
<a href="#">PEX Fieldsolver Endcap Spacing</a>	Specifies a virtual spacing between the source/drain boundary and an endcap for fieldsolver device extraction.
<a href="#">PEX Fieldsolver Mode</a>	Specifies the capacitance extraction accuracy mode used by the Calibre xACT tool.
<a href="#">PEX Fill Handling</a>	Specifies how floating nets are handled.
<a href="#">PEX Fracture Frequency</a>	Specifies the fracture frequency needed for node insertion.
<a href="#">PEX Ground Layer</a>	Specifies layers with shapes for ground regions.
<a href="#">PEX Ignore Capacitance</a>	Specifies to ignore certain types of capacitance between specific layers.
<a href="#">PEX Ignore Resistance</a>	Specifies to ignore resistance for specific layers or connections.
<a href="#">PEX Include Distributed</a>	Specifies to only include net names listed for input to distributed extraction.
<a href="#">PEX Include Lumped</a>	Specifies to limit lumped extraction to the listed net names. That is, only specified nets are processed.
<a href="#">PEX Indie Spacing</a>	Controls whether spacing for in-die variation is based on edges or density.
<a href="#">PEX Magnify</a>	Supports optical shrink for in-die variation processes.
<a href="#">PEX Netlist</a>	Generates a netlist with parasitic elements and places it in the file specified.
<a href="#">PEX Netlist ADMS</a>	Generates a netlist for the extracted block in an ADMS flow and stores it at the location specified.
<a href="#">PEX Netlist Character Map</a>	Specifies to replace characters in net names.
<a href="#">PEX Netlist Connection Section</a>	Specifies to include or omit * I and * P from DSPF and SPEF netlists.
<a href="#">PEX Netlist Create Smashed Device Names</a>	Specifies how to output smashed device instance names in DSPF and SPEF formats.
<a href="#">PEX Netlist Device Resistance Model</a>	Specifies which model to use for parasitic resistors.

**Table 3-13. Specification Statements: PEX (cont.)**

<b>Statement</b>	<b>Description</b>
<b>PEX Netlist Distributed</b>	Generates a netlist with distributed parasitic elements and stores it at the location specified.
<b>PEX Netlist Escape Characters</b>	Specifies escape character strings used in the output file.
<b>PEX Netlist Export Ports</b>	Specifies whether or not the formatter creates ports for internal nets.
<b>PEX Netlist Filter</b>	Specifies the removal of device instances from the extracted netlist.
<b>PEX Netlist Global Nets</b>	Removes the named nets from a subcircuit and places them in a global name definition.
<b>PEX Netlist Linewrap</b>	Specifies where the netlist formatter wraps the output line in the generated netlist.
<b>PEX Netlist LPE Ignore Idealnet</b>	Specifies whether or not the netlist formatter takes into account the nets that have been excluded from extraction when determining the worst case parasitic model.
<b>PEX Netlist LPE Using Extmode</b>	Instructs the netlist formatter to obtain LPE values from the mapfile based on the command line extraction mode.
<b>PEX Netlist Lumped</b>	Generates a netlist with lumped parasitic elements and stores it at the location specified.
<b>PEX Netlist Noxref Net Names</b>	Keeps or eliminates parasitic net models with _noxref from netlists.
<b>PEX Netlist Position File</b>	Generates a file containing device positions and orientation.
<b>PEX Netlist Replicated_Device Delimiter</b>	Specifies the delimiting character for replicated layout instances in the netlist.
<b>PEX Netlist SchematicOnly</b>	Generates a parasitic netlist that contains the devices and nets in the source netlist used for LVS.
<b>PEX Netlist Select File</b>	Specifies a file which is used for controlling the extracted netlist during the formatting stage.
<b>PEX Netlist Simple</b>	Generates a netlist for the extracted circuit without parasitic elements and stores it at the location specified.
<b>PEX Netlist Smashed_Device Delimiter</b>	Specifies the delimiting character for smashed or flattened layout instances in the layout netlist.
<b>PEX Netlist Subnode Section</b>	Specifies to include or omit * S from DSPF and SPEF netlists.
<b>PEX Netlist Unshort Device Pins</b>	Sets the value used for nominal resistors connecting shorted device pins.

**Table 3-13. Specification Statements: PEX (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">PEX Netlist Uppercase Keywords</a>	Specifies when to upper case element names and HSPICE and DSPF keywords in an output netlist.
<a href="#">PEX Netlist Virtual Connect</a>	Specifies how to handle disjoint net model fragments created by Virtual Connect statements.
<a href="#">PEX Pin Order</a>	Specifies source or layout pin ordering.
<a href="#">PEX Probe File</a>	Specifies specific points on a net to use for timing verification.
<a href="#">PEX Reduce CC</a>	Performs reduction of coupled capacitance on a net basis.
<a href="#">PEX Reduce Analog</a>	Specifies performing reduction of extracted data for analog designs.
<a href="#">PEX Reduce Digital</a>	Specifies to perform reduction of distributed extraction data such that there is no change in the time delay.
<a href="#">PEX Reduce Distributed</a>	Specifies to perform reduction of distributed RC extraction data in a way that has little impact on circuit characteristics such as delay up to a specified frequency.
<a href="#">PEX Reduce MinCap</a>	Specifies to combine or remove parasitic capacitors below a threshold value to reduce netlist size.
<a href="#">PEX Reduce MinRes</a>	Specifies to combine parasitic resistors below a threshold value to reduce netlist size.
<a href="#">PEX Reduce ROnly</a>	Performs reduction of resistance-only extracted data.
<a href="#">PEX Reduce TICER</a>	Specifies to perform reduction of distributed RC extraction data in a way that has little impact on circuit characteristics such as delay up to a specified frequency.
<a href="#">PEX Reduce Via Resistance</a>	Specifies how to handle clusters of identically-connected vias when calculating resistance.
<a href="#">PEX Report</a>	Specifies generating a report for parasitic results of the extracted circuit.
<a href="#">PEX Report Coupling Capacitance</a>	Generates a coupling capacitance-to-total capacitance report and stores it at the location specified.
<a href="#">PEX Report Distributed</a>	Generates a distributed parasitics report and stores it at the location specified.
<a href="#">PEX Report Lumped</a>	Generates a lumped parasitics report and stores it at the location specified.
<a href="#">PEX Report Mutual Inductance</a>	Generates a report of the mutual inductance on a per-net basis for victims specified by the victim file or victim SVRF statements and stores it at the location specified.

**Table 3-13. Specification Statements: PEX (cont.)**

<b>Statement</b>	<b>Description</b>
<a href="#">PEX Report Netsummary</a>	Generates a report of the total capacitances for a selected set of nets and stores it at the location specified.
<a href="#">PEX Report Point2Point</a>	Generates a report for point-to-point resistance of the specified nets.
<a href="#">PEX Resistance Parameters</a>	Specifies layer-specific parameters for resistance calculations.
<a href="#">PEX Sensitivity</a>	Generates a netlist with process variation sensitivity information.
<a href="#">PEX Slots Handling</a>	Specifies how to control extracting parasitics from slotted layers.
<a href="#">PEX Temperature</a>	Specifies a temperature dependency for the extracted resistance.
<a href="#">PEX Thickness EQN</a>	Specifies the thickness equation as a function of drawn width and local density.
<a href="#">PEX Thickness Nominal</a>	Specifies the nominal thickness for a layer.
<a href="#">PEX Threshold</a>	Specifies thresholds for distributed parasitic extraction such that if the specified threshold constraint is not met, lumped parasitic results are output instead.
<a href="#">PEX Tolerance Distributed</a>	Specifies to run reduction during distributed parasitic extraction such that the resistance and capacitance along the nets are preserved.
<a href="#">PEX Via Capacitance</a>	Specifies whether to include capacitance from vias and contacts.
<a href="#">PEX Via Reduction Resistance</a>	Specifies how to handle clusters of identically-connected vias when calculating resistance.
<a href="#">PEX Xcell</a>	Specifies a hierarchical extraction cell referred to as an xcell.
<a href="#">PEX Xcell Extract Mode</a>	Specifies how to handle geometries inside xcells when calculating parasitics.
<a href="#">PEX Xcell Precedence</a>	Specifies which file, either SVRF rule file or xcell list file, takes precedence if conflicting xcell definitions are encountered.
<a href="#">Resistance Connection</a>	Defines the proportionality constants for computing the resistance of current flowing between two conduction layers.
<a href="#">Resistance Device_Seed</a>	Defines the proportionality constants for computing the resistance of current flowing within a device seed layer.

**Table 3-13. Specification Statements: PEX (cont.)**

Statement	Description
<a href="#">Resistance Rho</a>	Defines the rho (bulk resistivity) of a layer, and is used for computing resistance variation due to in-die width and thickness variation.
<a href="#">Resistance Sheet</a>	Defines the proportionality constants for computing the resistance of current flowing within a conduction layer.

## PEX Inductance Specification Statements

[Table 3-14](#) lists the PEX inductance specification statements. These statements are used by Calibre xL to extract parasitic inductance.

**Table 3-14. Specification Statements: PEX Inductance**

Statement	Description
<a href="#">Inductance Wire</a>	Specifies the parameters to use for calculating the self inductance of bonding wires.
<a href="#">PEX Generate Driver File Tag</a>	Instructs the Calibre xRC tool to generate a driver and receiver file.
<a href="#">PEX Ground</a>	Specifies one or more ground net names to consider during inductance extraction.
<a href="#">PEX Inductance Default Partial Model</a>	Specifies whether to use partial inductance (PI) or per unit length (PUL) when calculating self impedance for paths that are 2 microns or longer and have no return path near them.
<a href="#">PEX Inductance Differential Pair</a>	Specifies the nets that are part of the differential pair and, optionally, the local return net.
<a href="#">PEX Inductance Driver File</a>	Specifies the location of the driver and receiver file.
<a href="#">PEX Inductance Driver Summary</a>	Specifies whether or not to output driver summary information to the transcript.
<a href="#">PEX Inductance Extract Layers</a>	Specifies to process only geometries on the specified layers during inductance extraction.
<a href="#">PEX Inductance Filter</a>	Specifies whether or not to filter paths for inductance extraction.
<a href="#">PEX Inductance Forward Coupling</a>	Specifies whether to turn forward coupling calculations on or off for self impedance.
<a href="#">PEX Inductance Frequency</a>	Specifies the global maximum operating frequency for inductance extraction for all signal nets in the design.

**Table 3-14. Specification Statements: PEX Inductance (cont.)**

Statement	Description
<b>PEX Inductance ... Frequency</b>	Specifies the nets for which the frequency is different from the global maximum operating frequency.
<b>PEX Inductance Layer Summary</b>	Specifies whether or not to output layer summary information to the transcript.
<b>PEX Inductance Minlength</b>	Specifies the minimum path length used to accurately compute self inductance.
<b>PEX Inductance Parameters</b>	Specifies the parameters to be used when calculating self inductance for large package vias.
<b>PEX Inductance Range</b>	Specifies how far to search from a given net segment for the return paths to include in self impedance calculations.
<b>PEX Inductance Returnpath</b>	Specifies the type of nets (ground or power) that may be considered as candidates for return path in inductance extraction.
<b>PEX Inductance Same Net Mutual</b>	Enables the extraction of the mutual inductance between wire segments within the same net, such as in stacked metal configuration.
<b>PEX Inductance Self</b>	Specifies the self inductance value to use on wire segments that are filtered.
<b>PEX Inductance Skin Include</b>	Specifies one or more nets to include in skin effect calculations.
<b>PEX Inductance Switch Time</b>	Specifies the global switch time to use for all nets in the design.
<b>PEX Inductance ... Switch Time</b>	Specifies the nets for which the switch time is different from the global switch time.
<b>PEX Inductance Victim</b>	Specifies a victim net name.
<b>PEX Inductance Victim Path</b>	Specifies the victim path associated with a victim net.
<b>PEX Inductance Victim File</b>	Specifies the location of the victim file.
<b>PEX Netlist Mutual Resistance</b>	Specifies whether or not to use current-controlled voltage sources to model mutual resistance.
<b>PEX Power</b>	Specifies one or more power net names to consider during inductance extraction.
<b>Unit Inductance</b>	Relates the preferred user-defined unit of inductance to the fixed standard unit of inductance, which is picohenries, for via and bonding wire inductance calculations.

## Unit Specification Statements

Table 3-15 lists the Unit specification statements. These specify the preferred units used in interpreting appropriate numeric values for capacitance, length, resistance, and time. These apply only to nmLVS and PEX.

**Table 3-15. Specification Statements: Unit**

Statement	Description
Unit Capacitance	Relates the preferred user-defined unit of capacitance to the fixed standard unit of capacitance, which is farads.
Unit Inductance	Relates the preferred user-defined unit of inductance to the fixed standard unit of inductance, which is picohenries, for via and bonding wire inductance calculations.
Unit Length	Relates the preferred user-defined unit of length to the fixed standard unit of length, which is meters. This statement has no effect on the rule file precision and is used only by nmLVS and PEX applications.
Unit Resistance	Relates the preferred user-defined unit of resistance to the fixed standard unit of resistance, which is ohms.
Unit Time	Relates the preferred user-defined unit of time to the fixed standard unit of time, which is seconds.

## Other Specification Statements

Table 3-16 lists the remaining specification statements that are not listed in the previous tables.

**Table 3-16. Specification Statements: Other**

Statement	Description
Include	Calls another rule file from the rule file in which this statement appears.
Net Area Ratio Print	Prints the input net area ratio accumulation (NARAC) information to a specified file.
SVRF Error	Issues a compiler error if this statement is encountered during rule file compilation.
SVRF Message	Issues a message in the run transcript when this statement is encountered during rule file compilation.
SVRF Version	Defines the earliest Calibre version that is permitted to compile the rule file.
Text Print Maximum	Controls the number of text objects and ports printed in the transcript of a Calibre run. Not used in ICverify.

**Table 3-16. Specification Statements: Other (cont.)**

Statement	Description
Title	Assigns a title to the rule file. The rule file title can be attached to DRC results in the nmDRC results database. It also appears in summary reports generated by the nmDRC application.
Variable	Defines the value of a variable in the rule file.

# Chapter 4

## Reference Dictionary

---

This chapter contains descriptions for the statements and operations that can be used in a rule file. These appear in alphabetical order.

Click a letter to skip to the first command beginning with that letter.

[A](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#)

## AND

Layer operation

Single layer syntax:

**AND *layer1* [*constraint*]**

Two-layer syntax:

**AND *layer2* *layer3* [[NOT] CONNECTED]**

### Parameters

- ***layer1***

A required original layer or layer set.

- ***layer2***

A required original layer or layer set, or a derived polygon layer.

- ***layer3***

A required original layer or layer set, or a derived polygon layer.

- ***constraint***

An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. Constraints must contain non-negative integers. The constraint limits the selection of polygon areas common to the number specified and defaults to  $> 1$ . The  $\text{== } 0$  constraint does not produce the complement of ***layer1***, but rather produces an empty derived layer.

- **[NOT] CONNECTED**

An optional keyword that specifies the electrical nodes of polygons are taken into account when determining the polygon intersections. The connectivity of ***layer2*** and ***layer3*** must be established to use this keyword. Possible choices are these:

CONNECTED — Specifies that polygons must have the same electrical node ID for their intersection to be output.

NOT CONNECTED — Specifies that polygons must have different electrical node IDs for their intersection to be output.

### Description

Constructs the intersection regions of polygons on the input layer(s) and outputs the intersections as polygons.

The single-layer Boolean AND operation handles overlapping polygons on the original input layer before pre-merging of the layer.

A single-layer operation can take a *constraint*. The *constraint* limits the construction of intersection regions to the number of polygons specified. For example, if  $> 2$  is specified, then more than two polygons must intersect in order for output to occur. Hierarchical Calibre applications perform the one-layer operation flat when the *constraint* is other than  $\geq 1$  or  $> 1$ .

## AND

---

The single-layer AND operation can give unexpected results in hierarchical operation for databases that have duplicate placements with overlapping geometry. Calibre selectively discards duplicate placements in many cases as part of its internal optimization. Databases with such characteristics are better handled using Pyxis Layout.

Both the single-layer and two-layer forms of the AND operation are node-preserving for connectivity extraction. For the two-layer form, connectivity is passed from the first input layer. See “[Node-Preserving Layer Operations](#)” in the *Calibre Verification User’s Manual* for details.

The [NOT] CONNECTED keywords are used only in the two-layer syntax. These keywords limit the determination of intersections based upon the presence (or lack, if NOT is used) of identical node IDs. Connectivity must be successfully established on the input layers for these keywords to be used.

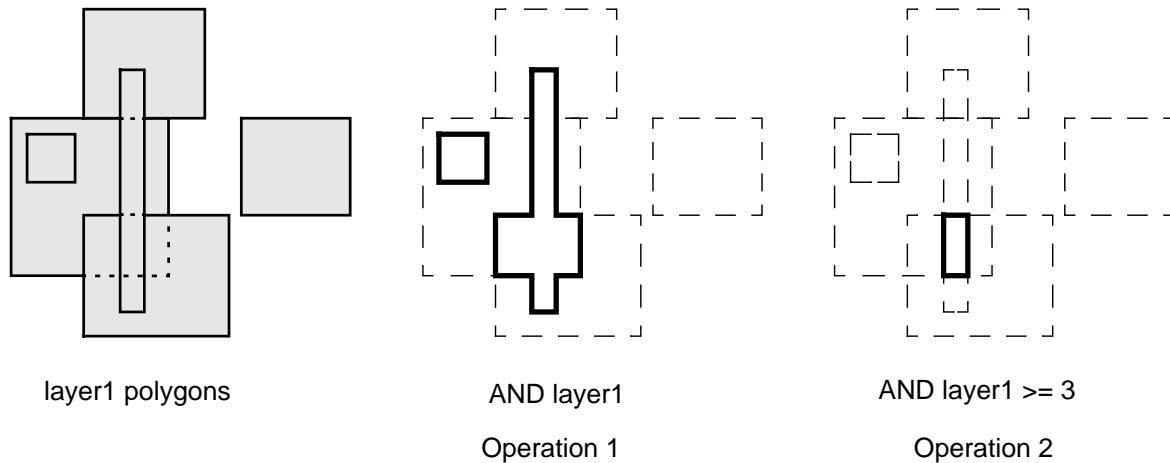
See also [NOT](#), [OR](#), [XOR](#).

## Examples

### Example 1

Figure 4-1 shows two examples of one-layer Boolean AND operations.

**Figure 4-1. One-layer Boolean AND Operation**



Operation 1 selects all areas on layer 1 common to more than one polygon; Operation 2 selects all areas on layer1 common to at least three polygons.

**Example 2**

The two-layer Boolean AND operation shown in Figure 4-2 selects all polygon areas common to both layer2 and layer3 polygons.

**Figure 4-2. Two-layer Boolean AND Operation**



If you rewrote the two-layer Boolean AND operation from Figure 4-2 as:

```
AND layer3 layer2
```

the operation does not produce a different geometric output because Boolean AND statements are commutative.

**Example 3**

The following is a common type of layer derivation:

```
gate = poly and diff
```

Note that the AND operation is node-preserving, so the gate layer inherits the connectivity of poly in this derivation. Ordinarily you would not want to derive gate like this:

```
gate = diff and poly // bad
```

This would cause gate to inherit the connectivity of diff, which is usually an error. However, this derivation would be geometrically identical to the previous one.

If the output layer appears in a Connect or Sconnect statement (but not as a BY layer), then connectivity is not passed by the AND operation.

**Example 4**

This example takes connectivity of the input layers into account. The intersections of the input layers are determined only if the polygons are not on the same node.

```
CONNECT m1 m2 BY via1
y = m1 AND m2 NOT CONNECTED // m1/m2 intersections on different nodes
z = via1 INSIDE y // vias inside the 2-node intersections
```

# Angle

Layer operation

## ANGLE layer constraint

### Parameters

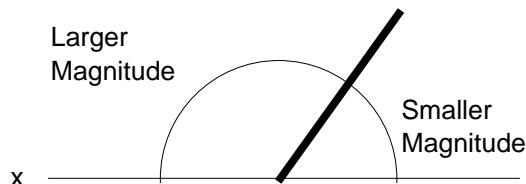
- ***layer***  
A required original layer or layer set, or a derived polygon or edge layer.
- ***constraint***  
A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. Constraints must contain floating-point numbers, interpreted in degrees, and must be greater than or equal to 0 but less than or equal to 90. You cannot specify the constraint  $> 90$ .

### Description

Selects all ***layer*** edges having a smaller angle magnitude with respect to the x-axis of the coordinate system, which conform to the ***constraint***.

Each edge has two angles in relation to the x-axis of the coordinate system: a smaller magnitude and a larger magnitude. Figure 4-3 shows both angles. The Angle operation always calculates the smaller angle magnitude between 0 and 90 degrees, inclusive.

**Figure 4-3. Angle Magnitudes in Relation to the X-axis**



Depending on your methodology, a narrow ***constraint*** range like  $> a < b$ , may be preferable to a constraint like  $\equiv a$ .

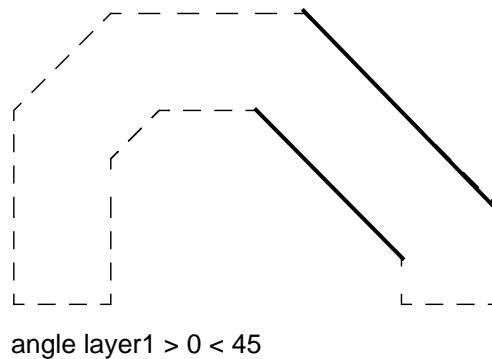
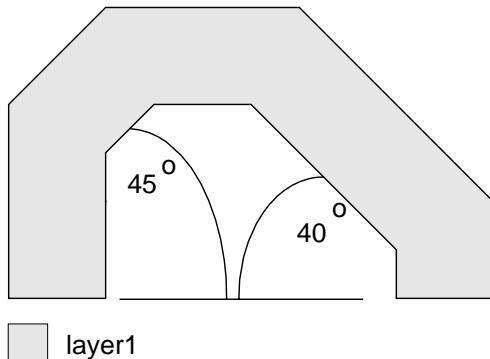
Hierarchical Calibre applications perform this operation hierarchically in all cases.

See also [Not Angle](#), [Drawn Acute](#), [Drawn Angled](#), [Flag Acute](#), [Flag Angled](#), [Exclude Acute](#), [Exclude Angled](#), and [Deangle](#).

## Examples

### Example 1

This operation selects all layer1 edges that conform to the constraint.



### Example 2

The following example of Angle operations selects all edges of layer metal that are not 45 degrees from, or orthogonal to, the x-axis:

```
metal_not_45 {
    // Check that all metal edges are either orthogonal or 45 degrees.
    ANGLE metal > 0 < 45
    ANGLE metal > 45 < 90
}
```

### Example 3

Sometimes it may be beneficial to use a narrow range for a constraint rather than a fixed value, depending on your methodology. For example:

```
ANGLE poly > 44.9 < 45.1
```

may be preferable to

```
ANGLE poly == 45
```

# Area

Layer operation

## AREA layer constraint

### Parameters

- ***layer***  
A required original layer or layer set, or a derived polygon layer.
- ***constraint***  
A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.  
The constraint is interpreted in user units squared.

### Description

Selects all ***layer*** polygons having areas that conform to the ***constraint***.

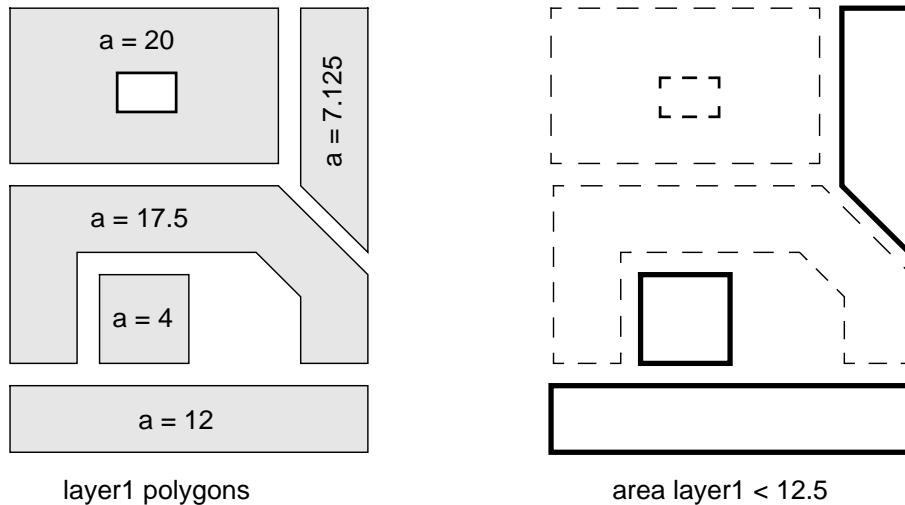
Depending on your methodology, a narrow constraint range like  $> a < b$ , may be preferable to a constraint like  $== a$ .

See also [Not Area](#).

### Examples

The Area operation shown in Figure 4-4 outputs all layer1 polygons with an area less than 12.5 user units squared.

**Figure 4-4. Area Operation**



# Attach

Connectivity extraction

**ATTACH *layer1* *layer2* [DIRECT] [MASK]**

## Parameters

- ***layer1***

A required original layer.

- ***layer2***

A required original layer or layer set, or a derived polygon layer. Must appear as an input layer to a [Connect](#) or [Sconnect](#) operation. In ICVerify, the Connect By or Sconnect operation must be in the same verification set as the Attach operation.

- **DIRECT**

An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.

- **MASK**

An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

## Description

The Attach operation is for assigning names to extracted nets or ports. It attaches connectivity information from a *layer1* object onto a *layer2* object.

Net labels that use the Attach statement are specified as layout database text objects for geometric databases (see [Text Layer](#)), or as [Layout Text \(File\)](#) labels specified in the rule file. If the *layer1* object is a text object, then only its basepoint must lie within *layer2*.

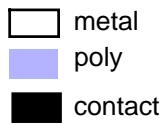
The poly\_txt object shown in Figure 4-5 overlaps the poly shape, and has a net name with a value of clock. The poly\_txt objects and poly shapes are on *layer1* and *layer2* of the Attach operation, respectively. Therefore, the Attach operation attaches the value of the net name owned by the poly\_txt object onto the poly shape.

**Figure 4-5. Attach Operation:**

CONNECT metal poly BY contact

TEXT LAYER poly\_txt

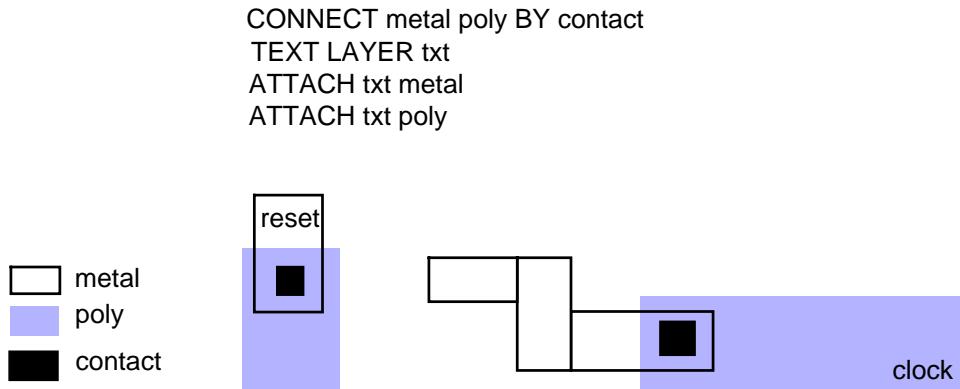
ATTACH poly\_txt poly



If *layer1* is used in more than one Attach operation, then each Attach operation attaches the values of the net names owned by the *layer1* objects onto the overlapping *layer2* objects.

In Figure 4-6 the text object overlapping the metal shape owns a net name with a value of reset, and the text object overlapping the poly shape owns a net name with a value of clock. The text objects are on *layer1* and the metal and poly shapes are on *layer2* in the respective Attach operations. Therefore, the Attach operations attach the net names reset and clock to the respective metal and poly shapes.

**Figure 4-6. Multiple Attach Operations**



If two or more *layer2* objects from different Attach operations overlap the same *layer1* object, then the Attach operation arbitrarily attaches the value of the net name owned by the *layer1* objects onto one of the *layer2* objects. A warning message is then issued. In addition, if no *layer2* objects overlap the *layer1* object, then the Attach operation ignores the value of the net name and issues a warning message.

The Attach statement attaches [Port Layer Text](#) objects and [Port Layer Polygon](#) shapes specified on *layer1* to *layer2*. Port names and locations are available to the layout verification application if it needs them.

If an attempt is made to attach a port to a non-connectivity layer, the Attach operation is ignored and a warning is issued. See “Transferring Logical Information on Merged Layers” in the [Calibre Verification User’s Manual](#) for detailed information on this subject.

**Requirements and Restrictions** —These restrictions apply when you use original layer sets as input layers to the connectivity extraction operations:

- A member of an original layer set that appears in an Attach operation cannot be used as an input layer to another Attach operation, or as an input layer to another connectivity extraction operation. For example, if met1\_txt is a member of the layer set all\_met\_txt, and all\_met\_txt appears in an Attach operation, met1\_txt cannot appear in a separate Attach or connectivity extraction operation.
- Any two, distinct, original layer sets used in one or more connectivity extraction operations in the same verification set cannot have layers in common. Which means the same layer member cannot appear in both distinct original layer sets.

## ICverify Considerations

This statement transfers the following in ICverify:

- Pin locations, port names, port locations, and associated must-connect information from Pyxis Layout shapes and paths on *layer1*, which are pin members, to overlapping polygons on *layer2*.
- Overflow locations from Pyxis Layout shapes and paths on *layer1*, which are connected by overflows, to overlapping polygons on *layer2*.

The tool reports ICverify ports on layers that do not appear in an Attach operation as unattached.

## Examples

```
// Original database layers are: poly, diffusion, metal,  
// contact_cut, text, and well.  
// Generate p and n source/drain regions for MOS transistors.  
sd = NOT diffusion poly  
nsd = AND sd well  
psd = NOT sd well  
  
// Establish connectivity.  
CONNECT nsd metal BY contact_cut  
CONNECT psd metal BY contact_cut  
CONNECT poly metal BY contact_cut  
  
// Attach diffusion information onto the source/drain interconnect layers.  
TEXT LAYER txt  
ATTACH txt metal  
ATTACH txt poly
```

# Capacitance Order

Parasitic extraction

**CAPACITANCE ORDER** *layer* [*layer* ...]

## Parameters

- *layer*

A required original layer or a derived polygon layer. If *layer* is not a base (substrate) layer, it must have connectivity. You can specify multiple different layers in one statement.

All layers that appear in capacitance calculations must appear in the Capacitance Order statement. The names and order must be consistent with the [Connect](#) statement.

Contact and via layers are not allowed in Capacitance Order statement. These layers break connectivity when performing capacitance extraction.

## Description

Defines the vertical order of the specified layers from bottom to top. For any shape where at least two conduction layers overlap, the Capacitance Order operation determines which layers are being shielded from both intrinsic and overlap capacitance effects with another layer.

There can be only one Capacitance Order operation in the rule file per connectivity mode. The ordering of layers in this operation is enforced as follows (where  $L_1 < L_2$  means that  $L_1$  is to the left of  $L_2$  in the operation, and  $L_1 > L_2$  means that  $L_1$  is to the right of  $L_2$  in the operation):

- For CAPACITANCE INTRINSIC A [B] INSIDE OF C:  $A < C$  and  $B < C$
- For CAPACITANCE CROSSOVER A B INSIDE OF C:  $(C < A \text{ and } C < B)$  or  $(C > A \text{ and } C > B)$
- For CAPACITANCE CROSSOVER A B INSIDE OF C D:  $C < A$  and  $C < B$  and  $D > A$  and  $D > B$
- For CAPACITANCE INTRINSIC A B ...: If B is in the Capacitance Order operation, then for any layer L in the Capacitance Order operation that is not a base layer in some Capacitance Intrinsic operation:  $B < L$ .

---

**Note**

If you are using coplanar layers, the Capacitance Order statement should be omitted from rule files generated by 2010.3 or later versions of xCalibrate.

---

## Examples

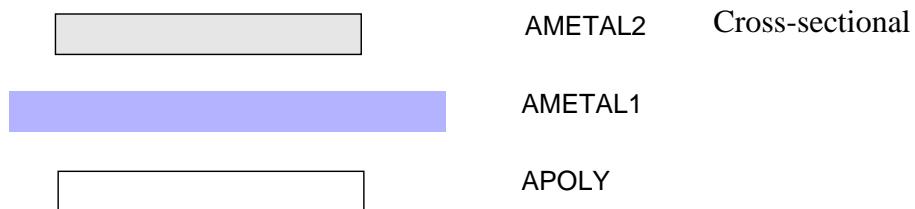
### Example 1

For a process stack such as the one shown in Figure 4-7, the equivalent Capacitance Order statement is

```
CAPACITANCE ORDER APOLY AMETAL1 AMETAL2
```

The layers APOLY, AMETAL1, and AMETAL2 are all conductors.

**Figure 4-7. Process Stack for Capacitance Order Example**



### Example 2

These statements show consistent ordering of layers:

```
CONNECT metal1 metal2 BY via  
CAPACITANCE INTRINSIC metal1 INSIDE OF metal2  
CAPACITANCE ORDER metal1 metal2
```

## Coincident Edge

Layer operation

**COIN**cident EDGE *layer1* *layer2*

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.

### Description

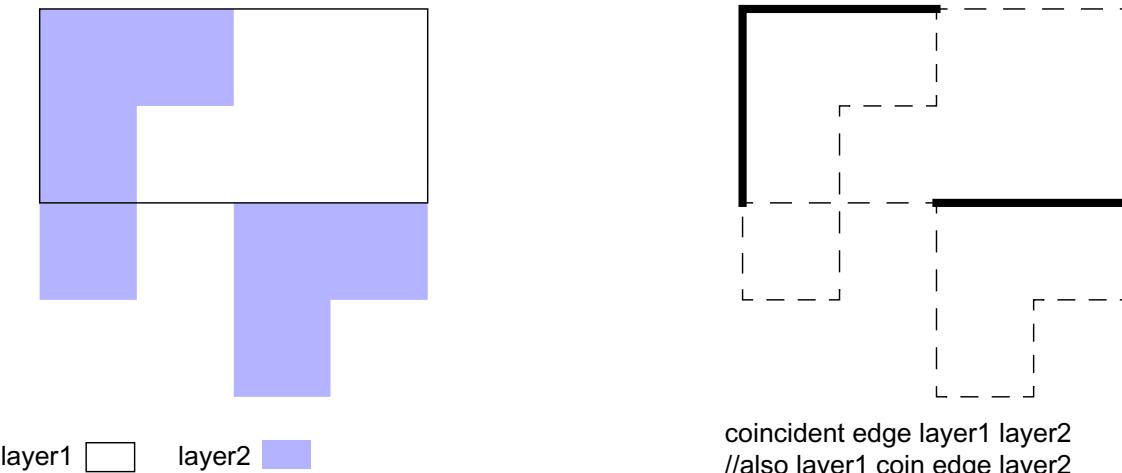
Selects all *layer1* edges or edge portions coincident with *layer2* edges.

See also [Not Coincident Edge](#), [Coincident Inside Edge](#), [Coincident Outside Edge](#), [Inside Edge](#), [Outside Edge](#), and [Touch Edge](#).

### Examples

The Coincident Edge operation shown in Figure 4-8 selects all layer1 edge segments that are coincident with layer2 edges.

**Figure 4-8. Coincident Edge**



## Coincident Inside Edge

Layer operation

**COINcident INside EDGE *layer1* *layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.

### Description

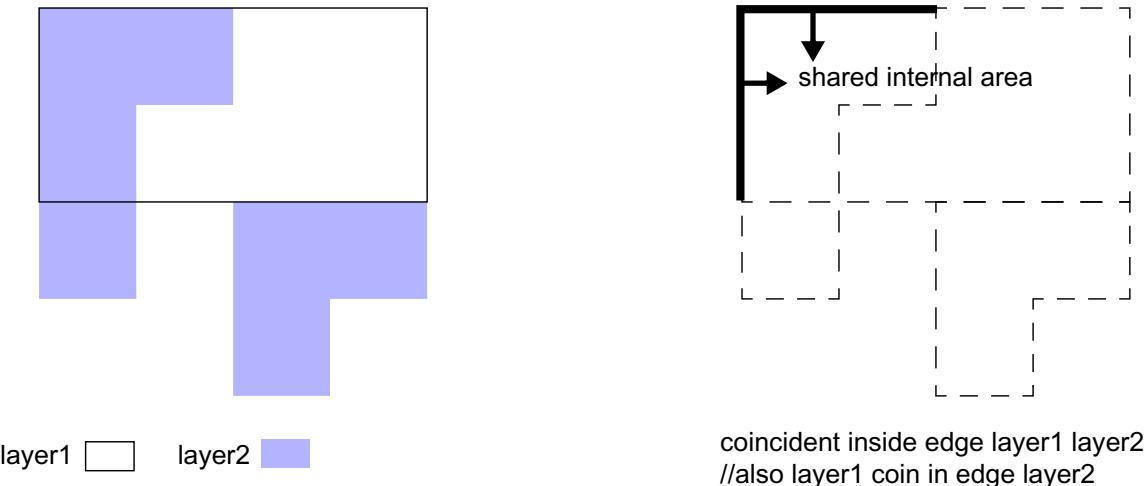
Selects all *layer1* edges or edge portions coincident with *layer2* edges, where the polygons share internal area.

See also [Not Coincident Inside Edge](#), [Coincident Edge](#), [Inside Edge](#), and [Touch Inside Edge](#).

### Examples

The Coincident Inside Edge operation shown in Figure 4-9 selects all layer1 edge segments that are inside-coincident with layer2 edges.

**Figure 4-9. Coincident Inside Edge**



## Coincident Outside Edge

Layer operation

**COINcident OUTside EDGE *layer1 layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.

### Description

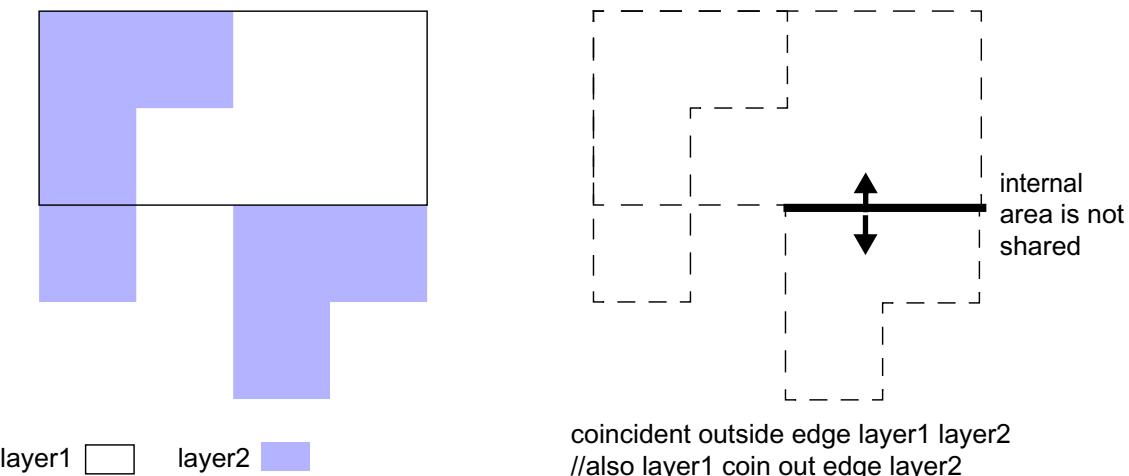
Selects all *layer1* edges or edge portions coincident with *layer2* edges, where the polygons do not share internal area.

See also [Not Coincident Outside Edge](#), [Coincident Edge](#), [Outside Edge](#), and [Touch Outside Edge](#).

### Examples

The Coincident Outside Edge operation shown in Figure 4-10 selects all *layer1* edge segments that are outside-coincident with *layer2* edges.

**Figure 4-10. Coincident Outside Edge**



# Connect

Connectivity extraction

Syntax 1:

**CONNECT *layer1* [*layerN* ...] [DIRECT] [MASK]**

Syntax 2:

**CONNECT *layer1* *layer2* [*layerN* ...] **BY** *layerC* [DIRECT] [MASK]**

## Summary

Establishes connectivity on specified layers by assigning nodal information to objects on the layers. Syntax 1 requires the polygons on the input layers to abut or overlap for connectivity to be established. Syntax 2 requires polygons from *layer1* and *layerC*, and the first available layer from *layer2* through *layerN* to intersect mutually for connectivity to be established.

## Parameters

- ***layer1***

A required original layer or layer set, or a derived polygon layer, followed optionally by other layers, layer sets, or derived polygon layers. You can have a maximum of 32 layers in the Connect operation. If this layer is a derived layer, it must be defined in the global scope of the rule file.

- ***layer2***

An original layer or layer set, or a derived polygon layer, followed optionally by other layers, layer sets, or derived polygon layers. This parameter is required in Syntax 2 but is not used in Syntax 1. You can have a maximum of 32 layers in the Connect BY operation. If this layer is a derived layer, it must be defined in the global scope of the rule file.

- **BY *layerC***

A required keyword for Syntax 2, followed by a required original layer or layer set, or a derived polygon layer. This specifies a contact layer. The secondary keyword BY must always precede the name of this contact layer when using this syntax. If you specify more than one *layerC* argument, the first is used and the rest are ignored. If this layer is a derived layer, it must be defined in the global scope of the rule file.

- **DIRECT**

An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.

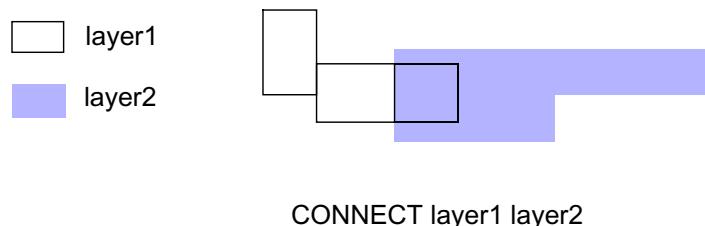
- **MASK**

An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

**Description**

A Syntax 1 Connect operation specifies electrical connection among *layer1* through *layerN* objects (typically shapes or paths) that abut or overlap, regardless of any other objects present on non-specified layers. For example, the *layer1* and *layer2* shapes shown in Figure 4-11 are all electrically connected.

**Figure 4-11. Connect**



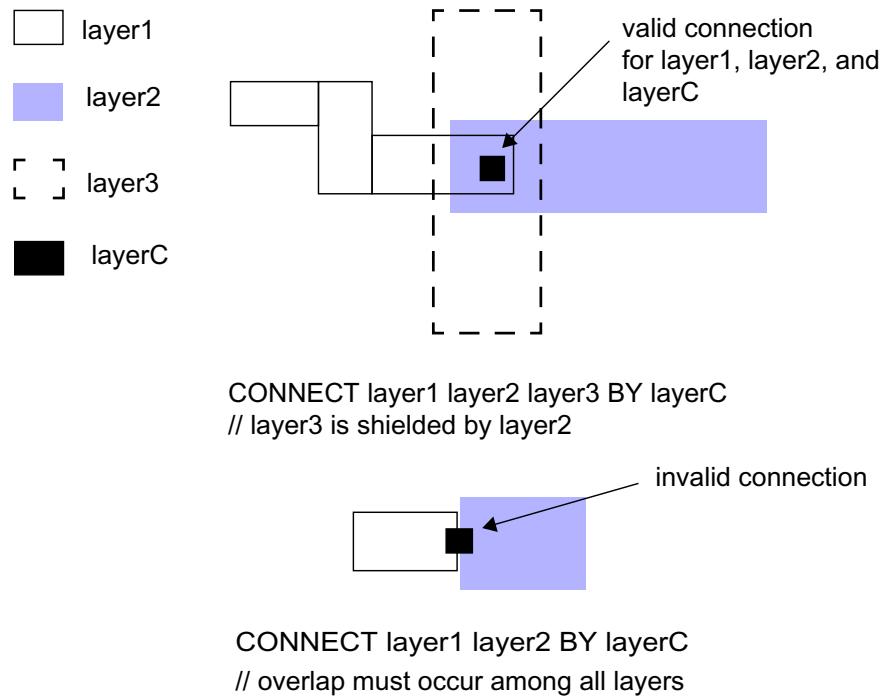
There are no restrictions on the ordering of *layer1* through *layerN* in a Connect operation using Syntax 1. For example, the following Connect operation is equivalent to the Connect operation shown in Figure 4-11.

```
CONNECT layer2 layer1
```

A Syntax 2 (Connect BY) operation specifies electrical connections among *layer1* and *layerC* objects, and the first available *layer2* through *layerN* object, in that order. The layers specified as *layer1* and *layerC* are always connected when they have mutual area overlap with one of *layer2* through *layerN*. The first object found from among *layer2* through *layerN* at a given location participates in the connection; no other layers from that set become connected.

For example, the *layer1* shape shown in Figure 4-12 is connected to the *layer2* shape by the *layerC* shape. However, *layer3* is specified after *layer2*, so *layer3* is not connected in this scheme due to shielding by *layer2*.

**Figure 4-12. Connect Operation Containing a Contact Layer**



If shielding occurs, hierarchical connectivity extraction must search the entire hierarchy to determine whether any layer2 objects are present. Objects whose connectivity cannot be unambiguously determined must be promoted out of a cell and up to a level of hierarchy where it can be determined. This can cause unexpected results such as the breaking of connectivity in hcells. There is also a performance penalty for using Connect BY statements that have shielding. For these reasons, you should generally avoid using Connect BY statements with more than *layer1* and *layer2*. For further discussion of this topic, refer to the “Shielding” section of the *Calibre Verification User’s Manual*.

To connect all three layers in Figure 4-12, you can specify a Connect BY statement for each layer pair, for example:

```
CONNECT layer1 layer2 BY layerC
CONNECT layer1 layer3 BY layerC
```

The connectivity model is built from all Connect and Sconnect statements in the rule file *after* all the derived layers required for the Connect and Sconnect statements have been generated. The connectivity model is constructed in a single step *before any operation that requires or passes connectivity*. (The only exception is when DRC Incremental Connect YES is specified.) For example:

```
CONNECT layer1
layer1 = layer2 AND layer3 // this layer is generated before connectivity
```

Here, the AND operation is performed before connectivity is established on layer1. If a node-preserving operation derives a layer that appears in a Connect or Sconnect statement, that operation does not pass connectivity.

This behavior can have unexpected side-effects if connectivity is not managed carefully. For example:

```
CONNECT layer1  
CONNECT layer2  
  
layer1 = layer2 AND layer3
```

Again, the AND operation is evaluated before connectivity is established. Node IDs are present on layer1 because of the CONNECT statement, but are not passed by the AND operation because there is no connectivity when this operation is performed (otherwise layer1 and layer2 would be shorted together). This can cause unexpected side-effects such as unattached label warnings involving connectivity layers and net names not being established in certain cases.

A good practice to follow is to use Connect statements for a relatively small set of layers. Then you can use node-preserving operations to pass connectivity to derived layers as needed. Careful management of derived layers appearing in connectivity statements is advised. Node-preserving operations are discussed in detail in the [Calibre Verification User's Manual](#).

Calibre nmDRC reports ports on layers that do not appear in a Connect operation as unattached.

**Requirements and Restrictions** — These restrictions apply when you use original layer sets as input layers to the connectivity extraction operations:

- You cannot use a member of an original layer set as an input layer to a Connect operation and as an input layer to another connectivity extraction operation. For example, if pwr\_met is a member of the layer set all\_met, and all\_met appears in an Connect operation, pwr\_met cannot appear in a separate Connect or connectivity extraction operation.
- Any two, distinct, original layer sets used in one or more connectivity extraction operations in the same verification set cannot contain common layers. Which means the same layer member cannot appear in both distinct original layer sets.

**Related statements** — [Sconnect](#) and [Stamp](#) are related connectivity operations. Sconnect provides unidirectional (one-way) transfer of connectivity information between layers. It is most often used in the detection of soft connections. Stamp is a layer operation that generates a derived layer that inherits the connectivity information of an input layer. Stamp also passes connectivity unidirectionally and is useful in situations where bidirectional passage of connectivity (such as with a Connect or Connect BY operation) is undesirable.

## Examples

```
psd = p_diff not gate
nsd = n_diff not gate
...
/* Establish circuit connectivity to the extent required by the design
rule checks.*/
CONNECT metall1 poly BY contact
CONNECT metall1 psd BY contact
CONNECT metall1 nsd BY contact
CONNECT metall1 metal2 BY via
...
PSD210 { EXT psd nsd < 4 NOT CONNECTED }
```

## Convex Edge

Layer operation

Basic syntax:

**CONVEX EDGE** *layer endpoint\_constraint* [WITH LENGTH *edge\_length\_constraint*]

Detailed syntax:

**CONVEX EDGE** *layer*

**ANGLE1** *angle\_constraint* [LENGTH1 *abut\_length\_constraint*]

**ANGLE2** *angle\_constraint* [LENGTH2 *abut\_length\_constraint*]

    [WITH LENGTH *edge\_length\_constraint*]

### Summary

Layer operation used to select edges based upon the number of convex endpoints the edges have. A convex endpoint is an endpoint where two edges join at a corner and the angle between the edges, measured through the polygon interior, is  $> 0$  and  $< 180$  degrees. Concave endpoints are selected by choosing appropriate constraint values.

### Parameters

- ***layer***

A required original layer, or a derived polygon or edge layer.

- ***endpoint\_constraint***

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. When used in the simple syntax, this constraint specifies the number of endpoints (0, 1, or 2), which must lie at a convex corner for an edge to be selected.

- **ANGLE1 *angle\_constraint* ANGLE2 *angle\_constraint***

Required keywords in the detailed syntax that define abutting edge behavior. You must specify **ANGLE1** and **ANGLE2** with their respective constraints, which are interpreted in degrees measured between interior-facing sides of abutting edges.

Constraints are listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

- **LENGTH1 *abut\_length\_constraint* LENGTH2 *abut\_length\_constraint***

Optional keywords in the detailed syntax that specify the edge selection depends on the length of abutting edges at the endpoints defined by **ANGLE1** and **ANGLE2**, respectively. The LENGTH1 and LENGTH2 keywords are specified with an associated *abut\_length\_constraint*. These constraints are interpreted in user units and apply to the edges that abut at the endpoints of a given edge.

Constraints are listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

- WITH LENGTH *edge\_length\_constraint*

An optional keyword in the detailed syntax that specifies that output edges are limited in length to *edge\_length\_constraint*. The length is in user units, governed by the “Constraint Notation” column of Table 2-2.

## Description

This operation has two syntaxes, basic or detailed. In the basic syntax, the operation creates a derived edge layer by selecting edges depending on the number of convex endpoints each edge has. Edge length, angle of abutting edges, and length of abutting edges are considered by using the detailed syntax.

This operation is not limited to selection of edges based purely on convexity. In most cases the convexity (or concavity) of the endpoints of edges is a by-product of the parameters you specify.

Both the basic and detailed specifications can include the WITH LENGTH optional keyword and the *edge\_length\_constraint*.

The differences between basic and detailed are discussed in the following sections.

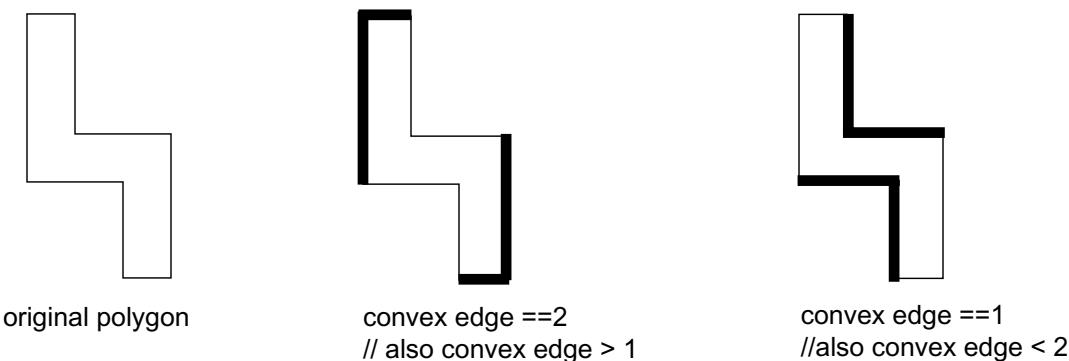
## Basic Endpoint Specification

A basic endpoint specification includes a constraint indicating the number of convex corner endpoints (0, 1, or 2) for the edge to be selected. A convex corner must have an interior measure of less than 180 degrees.

- To select edges with two endpoints at convex corners, use == 2.
- To select edges with exactly one endpoint at a convex corner, use == 1.
- To select edges with no endpoints at convex corners (both corners are concave), use == 0.
- To select edges with at least one endpoint at a convex corner, use > 0.
- To select edges that do not have both endpoints at convex corners, use < 2.

Example constraints are shown in Figure 4-13:

**Figure 4-13. Convex Edge Simple Specification**



### Detailed Endpoint Specification

A detailed endpoint specification allows edge selection based upon the exact angles formed by abutting edges at each endpoint of a given edge. **ANGLE1** designates one endpoint and **ANGLE2** designates the other endpoint. Angle constraints which are < 180 degrees (measured internal to the polygon) are convex. Angle constraints which are > 180 degrees are concave.

LENGTH1 and LENGTH2 further specify edge selection based on the length of the abutting edges at the endpoints of a given edge. Length parameters are associated with **ANGLE1** and **ANGLE2**, respectively. If you do not specify LENGTH1 and LENGTH2 they default to the constraint  $\geq 0$ .

Detailed endpoint specification selects edges with the following algorithm.

- Designate the endpoints of the edge as A and B.

The edge is selected if the status at endpoint A satisfies the **ANGLE1** and LENGTH1 constraints, and the status at endpoint B satisfies the **ANGLE2** and LENGTH2 constraints.

If the edge is not selected from the previous test, then:

- The edge is selected if the status at endpoint B satisfies the **ANGLE1** and LENGTH1 constraints and the status at endpoint A satisfies the **ANGLE2** and LENGTH2 constraints.

The tool does not select the edge if it does not meet the above requirements.

### Derived Edge Input

The process of edge selection needs to be refined for a derived edge input layer. This is due to the possibility that abutting edges may not be present.

For basic endpoint specification, the number of convex endpoints is counted as before. However, if an abutting edge is absent, the endpoint is not counted. This edge is then selected if the number of convex endpoints passes the associated *endpoint\_constraint*.

For detailed endpoint specification, the angle at an endpoint where there is no abutting edge is defined as 0 degrees. This value can be used since edges cannot meet at an angle of 0 degrees in a layer representing merged data. The length of the abutting edge (which is missing) is defined to be 0 user units. For example, if the **ANGLE1** constraint includes 0, an endpoint with no abutting edge satisfies the **ANGLE1** constraint.

## Examples

### Example 1

The following example selects all metal1 edges which have both endpoints at a convex corner:

```
x = CONVEX EDGE metal1 == 2
```

### Example 2

The following example selects all metal edges which have one endpoint at a convex corner and length less than 3 user units:

```
y = CONVEX EDGE metal == 1 WITH LENGTH < 3
```

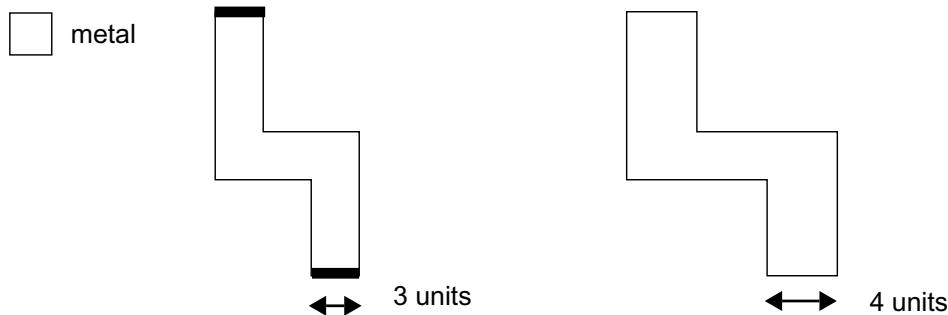
**Example 3**

The following example selects all edges from metal where one endpoint is at a convex corner with an abutting edge length of 3 or more, and the other endpoint is at a concave corner with an abutting edge length of any value.

```
z = CONVEX EDGE metal ANGLE1 < 180 LENGTH1 >= 3 ANGLE2 > 180
```

**Example 4**

The following example selects all edges from metal where both endpoints are at 90-degree convex corners and the edge itself has length greater than 2 and less than 4.



```
w = CONVEX EDGE metal ANGLE1 == 90 ANGLE2 == 90 WITH LENGTH >2 <4
```

**Example 5**

The following example selects all edges from metal where both endpoints are at 90-degree convex corners and at least one abutting edge has a length less than 3.

```
u = CONVEX EDGE metal ANGLE1 == 90 LENGTH1 < 3 ANGLE2 == 90
```

**Example 6**

The following example derives edge layers having gate edges inside diffusion, based upon the number of abutments with other gate edges inside diffusion.

```
gate_edges = poly INSIDE EDGE diff

// 0 abutments
gate_abut0 = CONVEX EDGE gate_edges ANGLE1 == 0 ANGLE2 == 0

// 1 abutment
gate_abut1 = CONVEX EDGE gate_edges ANGLE1 == 0 ANGLE2 > 0

// 2 abutments
gate_abut2 = CONVEX EDGE gate_edges ANGLE1 > 0 ANGLE2 > 0

// 0-1 abutments
gate_abut01 = CONVEX EDGE gate_edges ANGLE1 >= 0 ANGLE2 == 0

// 1-2 abutments
gate_abut12 = CONVEX EDGE gate_edges ANGLE1 >= 0 ANGLE2 > 0
```

## Copy

Layer operation

**COPY** *layer*

### Parameters

- *layer*

A required original layer or layer set, or a derived polygon or edge layer.

### Description

Copies the *layer* directly to a derived layer. The derived layer is the same type as *layer* unless *layer* is an original layer; then, the layer is a derived polygon layer.

Copy is a node-preserving operation. Layers derived from Copy operations generally have the same connectivity, if any, as the input *layer*. If a derived layer (in the global scope) from a Copy operation also appears in a [Connect](#) statement, or as the *lower\_layer* parameter of an [Sconnect](#) statement, then this does not occur. In such cases, the Copy operation has to be evaluated before the connectivity is established, so no connectivity is passed.

Copy has one historical behavior in this context that can be unexpected. Consider this example:

```
LAYER a 1
LAYER txt 2
TEXT LAYER txt

ATTACH txt b
CONNECT b
CONNECT c

b = COPY a // layers b and c have the same nodal information!
c = COPY a
```

Both layer b and layer c have *the same* node IDs, yet they appear in different Connect statements. This occurs because layers b and c are considered internally as aliases for the same layer. Hence, they have the same nodal information.

To avoid unexpected connectivity such as the preceding example discusses, you can do the following instead:

```
CONNECT b
CONNECT bb

b = COPY a // layers b and bb have different nodal information
bb = COPY b
```

Here, the copied layers do not have the same node IDs. This is probably the intended outcome.

See also [Flatten](#) and [Merge](#).

## Examples

### Example 1

The Copy operation is useful in debugging rule check statements and layer derivations. For example, consider this rule check statement:

```
metal_long_space {
  @ Spacing between metal edges longer than 100um must be at
  @ least 4um.
  long_metal = LENGTH metal > 100
  EXTERNAL long_metal < 4 }
```

During debugging of this rule check statement, you might desire to view the long\_metal layer. One way you can do this is to comment out the External operation and duplicate the Length operation as follows:

```
metal_long_space {
  @ Spacing between metal edges longer than 100um must be at
  @ least 4um.
  long_metal = LENGTH metal > 100
  LENGTH metal > 100
  // EXTERNAL long_metal < 4 }
```

However, the Copy operation makes this method more straightforward, as follows:

```
metal_long_space {
  @ Spacing between metal edges longer than 100um must be at
  @ least 4um.
  long_metal = LENGTH metal > 100
  COPY long_metal // For debugging.
  EXTERNAL long_metal < 4 }
```

### Example 2

You can use the Copy operation to debug global layer derivations. For example:

```
layer oxide 1
layer poly 2
inter_poly = poly NOT oxide
```

If you want to check what inter\_poly is visually, you can write a rule check:

```
debug {copy inter_poly}
```

### Example 3

The Copy operation enables you to easily attach multiple names to the same derived layer; for example:

```
X = INSIDE EDGE poly diff
Y = COPY X
```

The following works just as well, however:

```
X = INSIDE EDGE poly diff
Y = INSIDE EDGE poly diff
```

## **Copy**

---

### **Example 4**

The Copy operation is often used to derive layer copies in order to build up a new connectivity set from existing layers. For example:

```
CONNECT metall poly BY contact //original connectivity set  
  
// derive layer copies for new connectivity set  
c_metal1 = COPY metall  
c_contact = COPY contact  
c_poly = COPY poly  
  
CONNECT c_metal1 c_poly BY c_contact //special purpose connectivity set
```

The new connectivity set has different node IDs from the original one.

## Cut

Layer operation

**CUT** *layer1* *layer2* [*constraint* [BY NET] [EVEN | ODD]]

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.
- *constraint*  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. This constraint specifies the number of *layer2* polygons or nets that a *layer1* polygon must share some (but not all) of its area with to be selected by the Cut operation. The constraint must contain non-negative integers.
- BY NET  
An optional keyword used with the *constraint* parameter that specifies that a *layer1* polygon is selected when a number of distinct nets in the set of *layer2* polygons, which share some of their area with the *layer1* polygon, meets the specified *constraint*. The connectivity of *layer2* is required and is checked at compilation time.
- EVEN  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an even number. May not be used with ODD.
- ODD  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an odd number. May not be used with EVEN.

### Description

Selects all *layer1* polygons that share some (but not all) of their area with *layer2* polygons. Can also select all *layer1* polygons that share some (but not all) of their area with *layer2* polygons with a count that meets the given *constraint*.

If the interior of a *layer1* polygon lies completely inside a *layer2* polygon, and the polygons have coincident edges, this does not satisfy the Cut operation. Similarly, if the interior of a *layer1* polygon lies completely outside a *layer2* polygon, and the polygons have coincident edges, this does not satisfy the Cut operation.

If either input layer is empty, there is no output.

## BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that meet the constraint are output.

## EVEN and ODD Keywords

The EVEN and ODD keywords modify the interpretation of the *constraint*. For example:

**layer1 CUT layer2 >3 <9 EVEN**

This operation selects polygons from layer1 that share some, but not all, of their area with four, six, or eight polygons from layer2.

A constraint must be specified with these keywords. It is not useful to specify these keywords if your *constraint* does not allow for the number of polygons to meet the even or odd criterion, such as with == constraints.

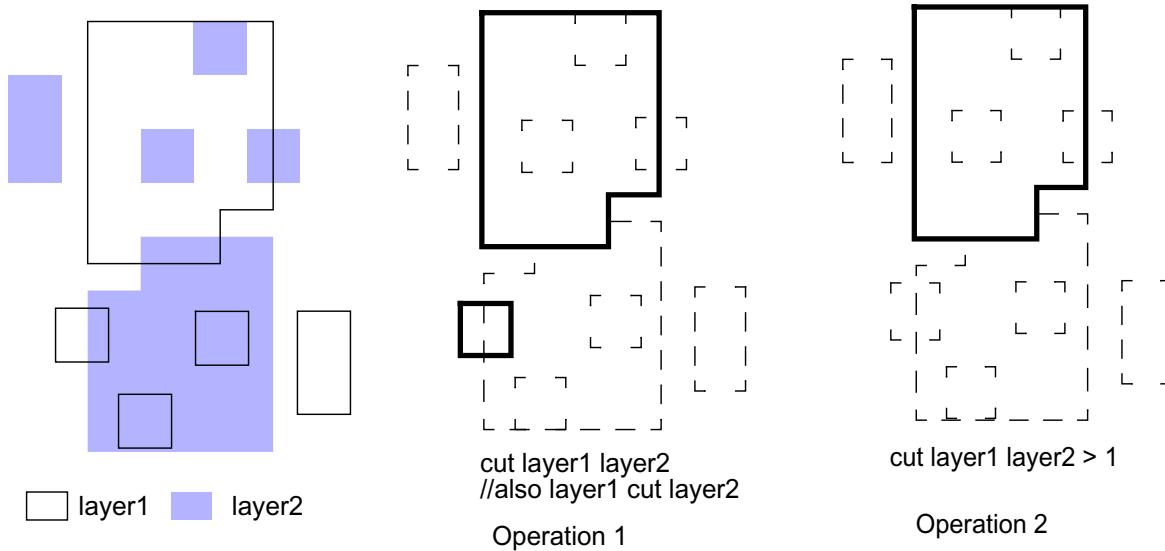
See also [Not Cut](#) and [Interact](#).

## Examples

### Example 1

[Figure 4-14](#) shows two Cut operations. Operation 1 selects all layer1 polygons that share some of their area with layer2 polygons. Operation 2 selects only the layer1 polygons that share some of their area with more than one layer2 polygon.

**Figure 4-14. Cut**

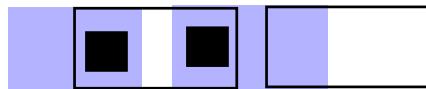


**Example 2**

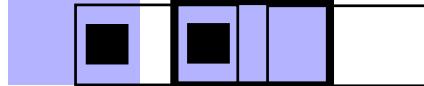
Figure 4-15 shows an example using the BY NET optional keyword.

**Figure 4-15. Cut BY NET**

- [Light Blue Box] pbase
- [Black Box] contact
- [White Box] nplus



connect pbase nplus by contact



cut pbase nplus >1 by net  
// find pbase polygons that cut  
// more than one net on nplus

## DBCLASSIFY

OPCverify operation

### DBCLASSIFY FILE

```
{['setup_commands '] | "filename" | inline_filename}
error_layer layer1 [... layerN] [MAP "UNIQUE ERRORS" | MAP "ALL ERRORS"] |
[MAP "XOR n"]
```

where *setup\_commands* are **HALO**, **CENTERPT**, and **TOLERANCE**, plus optionally one or more of the following:

**HALO** *halo\_value*

**CENTERPT** {**YES** | **NO**}

**TOLERANCE** *tol\_value* [*tol\_value2 tol\_value3...tol\_valuen*]

**MAXSIZE** *value*

FIRST LAYER IS CONTEXT

INSIDE LAST LAYER [XOR]

MAXIMUM EXACT CACHE SIZE *value*

MAXIMUM APPROXIMATE CACHE *value*

FLATTEN ONE PLACEMENT {**NO** | **YES**}

MATCH REFLECTIONS ROTATIONS {**YES** | **NO** | SYMMETRIC X-AXIS | SYMMETRIC Y-AXIS}

MAXIMUM ERROR NUMBER {**ALL** | *number*}

SUPPRESS [UNIQUE] [UNCLASSIFIED] [DUPLICATES]

MAXIMUM ERROR NUMBER {**ALL** | *number*}

CELL UNIQUE ERROR NUMBER CUTOFF [*number*]

**MAXTILESIZE** *value*

**TILINGMODE** {**ON** | **OFF**}

**PROPERTIES** {**YES** | **NO**}

**RDB** [ONLY] *filename*

### Summary

A Calibre OPCverify operation designed to output a meaningful subset of errors found with OPCverify checks by matching result markers on the derived error layer with relevant shapes (a relevant shape is a context clip around and under the error marker) on the input context layer(s). The settings you specify in the *setup\_commands* block control the sorting criteria.

## Parameters

- **FILE** {[ ‘*setup\_commands* ’] | “*filename*” | *inline\_filename*}

A required keyword that signals either the beginning of the setup commands block for DBCLASSIFY or an external file containing a setup file block to load. It takes one of the following parameters:

- [*setup\_commands*]

A group of setup commands on the command line for DBCLASSIFY, enclosed in square brackets ([ ]).

- “*filename*”

The pathname to a separate file that contains setup commands, which must be enclosed in quotes.

- *inline\_filename*

The name of a separately defined block of setup commands. You must specify the [Litho File](#) command using the supplied *inline\_filename* somewhere else in the SVRF rule file.

- ***error\_layer***

A required argument, specifying a LITHO layer (by name or number) that contains the error marker layer (polygons). You create this layer elsewhere in Calibre, usually with an SVRF rule check.

- ***layer1* [*layerN* ...]**

Keywords specifying one or more context layers from the layout. The first layer is required; additional layers are optional. Only the context layer(s) specified in this argument are considered when the configuration of polygons near the *error\_layer* markers are classified as matching or unique.

- **MAP “UNIQUE ERRORS”**

An optional argument; “UNIQUE ERRORS” must be enclosed by quotes. It generates a layer with just the unique errors. If MAP is not specified, this is the default behavior.

- **MAP “ALL ERRORS”**

An optional argument; “ALL ERRORS” must be enclosed by quotes. It generates a layer with all errors containing properties. The PROPERTIES YES option must also be used in the setup commands block if this option is specified.

- **MAP “XOR *n*”**

Optional argument that is used with the INSIDE LAST LAYER XOR keyword to handle concurrent output of the XOR layers. “XOR *n*” must be enclosed by quotes.

## Setup Commands

- **HALO *halo\_value***

A required argument specifying the radius of surrounding context, in microns, to include in the context search clip. It must be a value greater than or equal to 0.

- **CENTERPT {YES | NO}**

A required argument, specifying how to measure the halo radius.

- **YES**

DBCLASSIFY uses the centerpoint of the error shape and expands by the halo amount to create the context clip.

- **NO**

DBCLASSIFY uses the bounding box area of the error shape, then expands it by the halo amount to get the context clip.

- **TOLERANCE *tol\_value* [*tol\_value2 tol\_value3 ... tol\_valuen*]**

A required floating point value between 0 and 1, which represents the percentage of area required in an XOR operation between a context clip and a cached context clip. If optional additional context layers are specified, each context layer can have its own tolerance value set for it; if only one ***tol\_value*** is specified, that value is used for all layers.

---

**Note**

Specifying a tolerance of 0 may result in an apparent drop in unique errors versus non-zero tolerance unique error counts on the same design. This is due to the implementation of the cache size for exact matches versus the cache size for approximate matches; non-zero tolerance values may report unique errors in the unique error list more than once if the error expires from the cache. DBCLASSIFY finds all the unique errors regardless of the tolerance setting, however.

---

- **MAXSIZE *value***

An optional positive value. If the error shape bounding box is larger than MAXSIZE, then DBCLASSIFY writes the error as a unique error (*rulename\_uncl* group) without bothering to cache it, or to look it up in the cache. This option is only allowed when the CENTERPT option is specified as “no.”

- **FIRST LAYER IS CONTEXT**

An optional keyword that causes the first input layer (the *error\_layer*) to DBCLASSIFY to also be used as a context layer.

- **INSIDE LAST LAYER [XOR]**

An optional keyword that causes the final input layer (*layerN*) to be used as a filter marker layer; only shapes inside the markers on the last layer are classified. The unique node number of the inside layer shapes is used as a context for determining if an error shape is unique.

If XOR is specified with this keyword, the XOR'd context differences between the unique shapes within each inside-layer shape can be written out as concurrent layers. You can output the concurrent layers with the MAP “UNIQUE ERRORS” and MAP “XOR *n*” commands.

- MAXIMUM APPROXIMATE CACHE SIZE *value*

An optional keyword that specifies the maximum cache size for approximate matches. The approximate cache size does not drop below the specified value. The default value is 200 matches. Note that the approximate cache lookup algorithm is a linear search; increasing this value much larger than 200 will severely impact its performance, but the maximum value is MAX\_INT.

- MAXIMUM EXACT CACHE SIZE *value*

An optional keyword that specifies the maximum cache size for exact matches. When this number is exceeded, the cache size is reduced to size ( $4*value/5$ ). The default value is 50000 matches.

- FLATTEN ONE PLACEMENT {NO | YES}

An optional keyword used in hierarchical processing.

In a hierarchical design, DBCLASSIFY may find a unique error shape stored in a cell that has many placements in the top cell of the design.

Specifying YES for this keyword causes one instance of this error shape to be flattened to the top cell, and the in-cell error shape is deleted. The flattened shape is located at the lowest, leftmost placement of the cell transformed to-world to the top cell.

The default setting for this keyword is NO.

This keyword has no effect in flat Calibre runs.

**Note**

The SVRF rule file must also include a **DRC Check Map** command associated with the derived layer for this command to send the output to Calibre WORKbench.

- MATCH REFLECTIONS ROTATIONS {YES | NO | SYMMETRIC X-AXIS | SYMMETRIC Y-AXIS}

An optional keyword used in hierarchical processing.

In a hierarchical design, DBCLASSIFY may find a cell containing unique error shapes that have placements in the top cell that are mirrored and/or rotated.

For asymmetric mask post-processing (due to asymmetrical illumination), multiple placements of a cell with different orientations are not identical at the wafer.

- YES is the default if this keyword set is not specified.
- NO forces different orientations of identical contexts not to match. It also causes Calibre to clone cell transforms that are mirrored and/or rotated. This can cause

some additional hierarchical processing time within a Calibre run due to the extra cells created.

- SYMMETRIC X-AXIS causes identical contexts that are mirrored across the X axis to match.
  - SYMMETRIC Y-AXIS causes identical contexts that are mirrored across the Y axis to match.
- SUPPRESS [UNIQUE] [UNCLASSIFIED] [DUPLICATES]

An optional keyword that suppresses output of the specified type(s). You would primarily use this option to reduce the size of RDB files.

- MAXIMUM ERROR NUMBER {ALL | *number*}

An optional argument, specifying the maximum *number* of hierarchical error shapes that DBCLASSIFY counts before it skips the classification step; all error shapes are then categorized as unclassified in the RDB.

By default, DBCLASSIFY tries to classify all error shapes, regardless of number.

- CELL UNIQUE ERROR NUMBER CUTOFF *number*

An optional performance parameter that limits the number of tile unique contexts that are compared within a cell, and the number of cell unique contexts that are compared to other cells in the design. Once the specified cutoff limit is reached, the comparison routine stops, and all remaining uncompered errors are reported as unique.

This option is useful in the following situations:

- There are a very large number of contexts ( $> 1,000,000$ ) to be compared. The global identification of duplicate contexts can be a bottleneck on the master CPU and can severely limit the scalability of MTFLEX runs.
- For approximate match cases (TOLERANCE  $> 0$ ).

A good starting value for this parameter is about 100,000 (twice the size of the exact match cache size). Note that setting this parameter might cause some duplicate contexts to be incorrectly classified as unique contexts.

If CELL UNIQUE ERROR NUMBER *cutoff* is not specified, the value of LONG\_MAX is used as the cutoff value.

- MAXTILESIZE *value*

An optional keyword and parameter. If specified, this option can cause more/smaller tiles to be created. Use this parameter in cases where the halos of the error shapes are large enough to overlap each other, causing contexts with a very large number of edges to be generated. Forcing more tiles in this case may reduce the occurrence of a large tile tying up a slave. The smallest allowable specified value is 1; however the number actually used is the minimum tile fringe necessary to cover any error shape halo in a particular cell. Requires TILINGMODE to be set to ON.

- TILINGMODE {ON | OFF}

A performance parameter that toggles whether or not large cells are broken up into smaller tiles. The default value is OFF (does not break up tiles).

- PROPERTIES {YES | NO}

An optional argument that filters the markers to only errors that have properties attached. The default is to return all errors.

- RDB [ONLY] *filename*

An optional argument, specifying the name of an ASCII results database format file. DBCLASSIFY writes the file as the result of its operation. It contains the following RULECHECKS:

- *rulename\_uniq* — For unique errors.
- *rulename\_uncl* — For unclassified errors.
- *rulename\_dupl* — For duplicate errors.

where *name* is the layer name specified in the LITHO OPCVERIFY MAP argument, or the rule name if DBCLASSIFY was run as a standalone command.

This ASCII database contains errors from each of the rule checks. Errors use the conventions shown in [Table 4-1](#).

**Table 4-1. Rule Check Properties**

Property	Description
CLASS_ID <i>num</i>	<ul style="list-style-type: none"> <li>• <i>rulename_uniq</i>: Contains unique integers for each individual error.</li> <li>• <i>rulename_dupl</i>: Contains a non-unique number that refers to the unique error in <i>rulename_uniq</i> corresponding to this error. This provides a capability to associate representative unique errors to all errors whose class it represents.</li> </ul>
DUP_COUNT <i>num</i>	<ul style="list-style-type: none"> <li>• <i>rulename_uniq</i>: Contains the count of all the errors that occurred for this unique context error.</li> </ul>

**Table 4-1. Rule Check Properties (cont.)**

Property	Description
PLACE_COUNT <i>num</i>	<p>Is the number of placements of the error in the design with respect to the top cell. The PLACE_COUNT property is written only for calibre -hier runs.</p> <ul style="list-style-type: none"> <li>• <i>rulename_uniq</i>: Contains the placement count of all the errors that occurred for this unique context error. If the design is flat, the DUP_COUNT and PLACE_COUNT numbers will be the same. The DUP_COUNT number may be less than the PLACE_COUNT number if the design is hierarchical and you run Calibre in -hier mode.</li> <li>• <i>rulename_dupl</i>: Contains the number of placements of this duplicate error in the hierarchical design. <ul style="list-style-type: none"> <li>• If the design is flat, this number will be 1.</li> <li>• If the design is hierarchical, but all the error shapes are in the top cell, then the PLACE_COUNT number for duplicate errors will also be 1.</li> </ul> </li> </ul>

Specifying the optional ONLY switch causes DBCLASSIFY to only write out an RDB database, instead of also sending output to a layer.

### Description

DBCLASSIFY is an SVRF command that requires one or more “context layers” and an “error layer” or locale layer, all of which are polygon layers. It hierarchically scans all errors in terms of the context layers, and classifies (partitions) the errors based on the context layer configurations within the error’s vicinity.

- DBCLASSIFY uses each polygon on the error layer as the coordinates for the centerpoint of its halo classification (see the HALO and CENTERPT commands) and reporting.
- The *error\_layer* is not used as part of the context classification (unless FIRST LAYER IS CONTEXT is also specified); if the shapes of the errors are different but the contents of the given context layers are the same, then the error shapes are classified under the same context. If the shapes of the simulated error are to be considered as part of the context, then the layer containing those shapes should also be included in the context layer argument list.

Only one error for each unique context instance is returned on the output layer; each error and its context layer shapes are iteratively XOR-compared with previously stored context clips. If  $(\text{area\_xor}) / (\text{area\_shape}) < \text{tol\_value}$ , then DBCLASSIFY considers this clip as matching the cached context clip. Otherwise, it will become a unique new error.

#### Note



DBCLASSIFY is one of the commands Calibre OPCverify runs in Verification Center (described in the [OPCverify User and Reference Manual](#)).

DBCLASSIFY can return an ASCII RDB database that can be loaded into a results viewer to support cross-referencing capability.

## Notes

- Calibre LITHO may return a warning message about two duplicate layers being specified in the layer argument list. To work around this problem, make a copy of the duplicate layer and use it to avoid the duplicate layer error message.
- When used with hierarchical input, DBCLASSIFY may report fewer duplicate errors than are actually in the list, due to the nested storage method used to store duplicate errors in the hierarchy. Flattening the error layer before passing it to DBCLASSIFY will cause all duplicate errors to be reported at the top cell level, but using this method to work around this problem is not recommended due to the potential for very large memory and CPU run time.
- When running the calibre executable with the -turboflex option, adding or referencing a new layer in your SVRF rule file (such as adding a new DRC CHECK MAP statement) may result in a different number of approximate errors. The new layer modifies the database extent, resulting in the HDB building a different cell hierarchy. The modified cell hierarchy can re-order the selection of which errors are used as dictionary reference to compare to possible errors.

For example, error X and Y are different by 0.2, Y and Z are different by 0.2, and X and Z are different by 0.4. If the approximate match tolerance is set to 0.3 and X is encountered first and used as the reference context, DBCLASSIFY returns two approximate uniques (X matches Y, but not Z). If Y is encountered first and used as the reference context, DBCLASSIFY returns three approximate uniques (Y matches X and Y matches Z). This is expected behavior; Mentor Graphics does not guarantee a minimum number of unique errors.

## Examples

### Example 1

Does an exact match, suppresses the unique category in RDB, makes matching sensitive to orientation of context shape environment, and returns only one flattened instance of a unique nested shape that may have multiple placement counts.

```
ExactResult_UNIQ { dbclassify FILE [
    HALO 3
    CENTERPT no
    TOLERANCE 0.0
    MAXSIZE 200.0
    SUPPRESS UNIQUE
    MATCH REFLECTIONS ROTATIONS NO
    FLATTEN ONE PLACEMENT YES
    RDB ./dbclassifyexample.rdb
] LOCALE CONTEXT1}
```

**Example 2**

Returns approximate matches, suppressing the unique category in the RDB. Matches reflections and rotations. Does not flatten hierarchical designs:

```
ApproxResult_UNIQ { dbclassify FILE [
    HALO 3 CENTERPT YES
    TOLERANCE 0.2
    MATCH REFLECTIONS ROTATIONS YES
    FLATTEN ONE PLACEMENT NO
    SUPPRESS UNIQUE
    RDB ./dbclassifyexample.rdb
] LOCALE CONTEXT1}
```

**Example 3**

Suppresses output of duplicate errors, exact match:

```
DUPL_SUPPRESSED = dbclassify FILE [
    HALO 3 CENTERPT no TOLERANCE 0.0 MAXSIZE 200.0
    SUPPRESS DUPLICATES
    RDB ./dbclassifyexample.rdb
] LOCALE CONTEXT1
ExactResult_DUPL {COPY DUPL_SUPPRESSED}
```

**Example 4**

Suppresses output of duplicate errors, approximate matches

```
ApproxResult_DUPL { dbclassify FILE [
    HALO 3
    CENTERPT no
    TOLERANCE 0.001
    MAXSIZE 200.0
    SUPPRESS DUPLICATES
    RDB ./dbclassifyexample.rdb
] LOCALE CONTEXT1 CONTEXT2}
```

**Example 5**

Does not write the RDB file (no RDB statement in this code):

```
ExactResult_NORDB { dbclassify FILE [
    HALO 3
    CENTERPT no
    TOLERANCE 0.0
    MAXSIZE 200.0
] LOCALE CONTEXT1}
```

**Example 6**

Does not write a layer (only an RDB file is written):

```
ApproxResult_RDBONLY { dbclassify FILE [
    HALO 3
    CENTERPT no
    TOLERANCE 0.001
    MAXSIZE 200.0
    SUPPRESS DUPLICATES UNCLASSIFIED UNIQUE
    RDB ONLY ./dbclassifyexample.rdb
] LOCALE CONTEXT1}
```

**Example 7**

Calls a separately-defined [Litho File](#) block:

```
LITHO FILE classify_1 [ /*  
    HALO 0  
    CENTERPT no  
    TOLERANCE 0  
    MAXSIZE 15.0  
    PROPERTIES YES  
    RDB ./test.rdb  
*/ ]  
  
errchk1 = DBCLASSIFY FILE classify_1 ERRLAYER CONTEXT1 MAP "errchk1"  
errchk1 { COPY errchk1 } DRC CHECK MAP errchk1 101
```

**Example 8**

One or more XOR context layers to filter output layers can be sent to the output file as concurrent layers by creating a DBCLASSIFY block with the following MAP configuration:

```
uniques {dbclassify FILE error_layer layer1 context1 context2 context3  
    filter MAP "UNIQUE ERRORS" } DRC CHECK MAP uniques 110  
xor_context1 {dbclassify FILE error_layer layer1 context1 context2  
    context3 filter MAP "XOR 1" } DRC CHECK MAP xor_context1 111  
xor_context2 {dbclassify FILE error_layer layer1 context1 context2  
    context3 filter MAP "XOR 2" } DRC CHECK MAP xor_context2 112  
xor_context3 {dbclassify FILE error_layer layer1 context1 context2  
    context3 filter MAP "XOR 3" } DRC CHECK MAP xor_context3 113
```

The MAP “UNIQUE ERRORS” keyword is required to start the block; each xor\_n layer in subsequent MAP “XOR n” statements corresponds to the relevant context layer in the list.

## Deangle

Layer operation

**DEANGLE** *layer* **maximum\_deviation** [{SKEW [ORTHOGONAL ONLY]} | ALL]

Used only in Calibre applications.

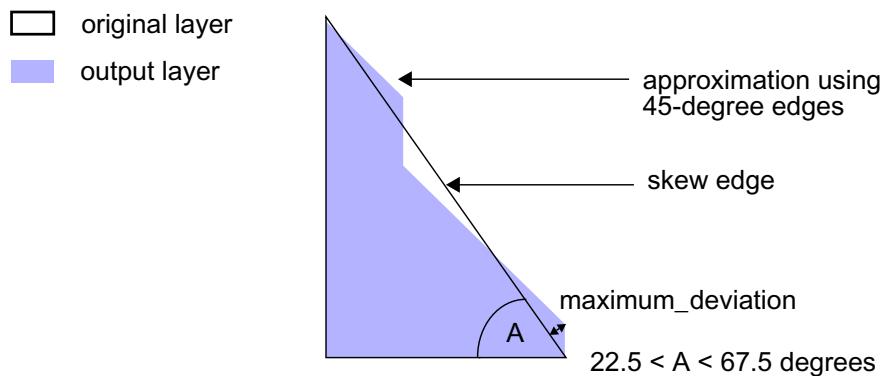
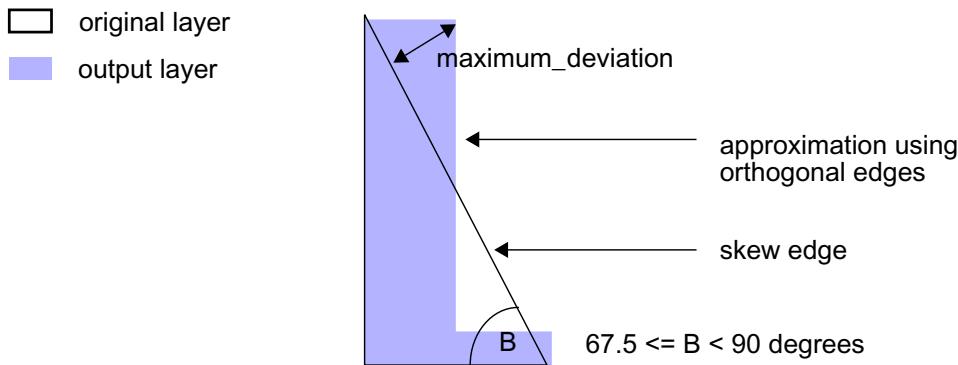
### Parameters

- ***layer***  
A required original layer or layer set, or a derived polygon layer.
- ***maximum\_deviation***  
A positive, floating-point numeric in user units and must be an integer when multiplied by the process precision.
- **SKEW**  
An optional keyword that approximates skew edges with combinations of orthogonal and 45-degree edges. This is the default behavior if no keyword from this set is specified.
- **ORTHOGONAL ONLY**  
An optional keyword that approximates skew edges with orthogonal edges only. This keyword must be used in conjunction with SKEW.
- **ALL**  
An optional keyword that approximates any angled edge with orthogonal edges, regardless of its angle with respect to the database axes.

### Description

Creates a derived polygon layer by replacing skew (not orthogonal or at a 45-degree angle with respect to the database axes) edges from *layer* with sequences of orthogonal or 45-degree edges. For the remainder of this discussion, an orthogonal edge is one that is orthogonal to the database axes. The operation is a non-node-preserving layer constructor. If *layer* is empty, the output layer is empty.

If either SKEW or no optional keyword is specified, the operation proceeds as follows. If *layer* has no skew edges, then the output layer is a copy of *layer*. Otherwise, skew edges of polygons on the input layer are replaced with one or more 45-degree or orthogonal edges. Edge replacement occurs both on the inside and outside of the original skew edge. The ***maximum\_deviation*** parameter describes the maximum perpendicular distance from the original skew edge to any point on any new 45-degree or orthogonal edge. See [Figure 4-16](#) and [Figure 4-17](#).

**Figure 4-16. Deangle Output Using 45-Degree Edges****Figure 4-17. Deangle Output Using Orthogonal Edges**

Skew edges forming an angle  $A$  with the horizontal such that  $22.5 < A < 67.5$  degrees are approximated using 45-degree edges. Skew edges forming an angle  $B$  with the horizontal such that  $0 < B \leq 22.5$  degrees, or  $67.5 \leq B < 90$  degrees, are approximated using orthogonal edges.

If SKEW is specified with ORTHOGONAL ONLY, then only orthogonal edges are used in the approximation, regardless of the angle of the skew edge with the horizontal.

If ALL is specified, then the operation proceeds as follows. If the input layer has no angled edges (see [Drawn Angled](#) for the definition), then the output layer is simply a copy of the input layer. Otherwise, angled edges of polygons on the input layer are replaced with one or more orthogonal edges. Edge replacement occurs both on the inside and outside of the original angled edge. The **maximum\_deviation** parameter describes the maximum perpendicular distance from the original edge to any point on any newly-constructed orthogonal edge.

## Examples

```
// approximate skew edges on layer1 with orthogonal edges using a
// maximum deviation of 0.1
ortho = DEANGLE layer1 0.1 SKEW ORTHOGONAL ONLY
```

# Density

Layer operation

**DENSITY** *layer1* [*layerN* ...] [‘[*density\_expression* ‘]’] **constraint**  
 [WINDOW {*wxy* | *wx wy*} [STEP {*sxy* | *sx sy*}]]  
[TRUNCATE | BACKUP | IGNORE | WRAP]  
 [INSIDE OF EXTENT |  
 INSIDE OF *x1 y1 x2 y2* |  
 {INSIDE OF LAYER *layerB* [BY EXTENT [*size\_value*] | BY POLYGON | BY  
 RECTANGLE | CENTERED *value*]})]  
 {[GRADIENT *constraint* [RELATIVE | ABSOLUTE] [CORNER [*value*]]] |  
 [MAGNITUDE *constraint* [RELATIVE | ABSOLUTE]]}]  
 [CENTERS *value*]  
 [PRINT [ONLY] *filename*]  
 [RDB [ONLY] *filename* [MAG *value*]  
 [COMBINE *constraint* [RELATIVE | ABSOLUTE]]]

## Summary

The Density operation is typically used to compute the density of an input layer within a specified data capture window, such as in this design rule: “The density of metal2 in every  $50 \times 50$  area of the layout must exceed 25%.” The default Density function is defined as the ratio of the total area of the input layers within the data capture window to the area of the window itself. The density is calculated as the window is tiled over a specified data capture extent. Each window placement that meets the constraint value is output as part of a merged layer.

The size of the data capture window, the data capture extent, and how the data capture window is tiled across the capture extent are controlled with options. The *density\_expression* option is used to provide a user-defined calculation instead of the default density calculation. Other options control the format of the results output, including detailed RDB output for statistical analysis.

Also see “[Density Best Practices](#)” in the *Calibre Solutions for Physical Verification* manual.

## Parameters

- ***layer1***  
 One or more original or derived polygon layers. At least one layer must be specified.
- ‘[*density\_expression* ‘]’  
 An optional expression enclosed in square brackets that performs a calculation involving layer data. If no *density\_expression* is provided, then the density is calculated as the ratio of the area of the input layers in a data capture window to the area of the data capture window itself. If you supply a *density\_expression*, then the value (based in user units) of the *density\_expression* is computed within each data capture window. If the value of the expression satisfies the **constraint**, then the window is output as usual.

The *density\_expression* may involve numbers (including numeric variables), operators, parentheses ( ), and the functions given in [Table 4-2](#):

**Table 4-2. Density Math Functions**

Function	Definition
AREA( <i>input_layer</i> )	Area of <i>input_layer</i> in user units of length squared. AREA() gives the area of the data capture window.
SQRT(x)	square root of x
EXP(x)	exponential (base e) of x
LOG(x)	natural logarithm of x
SIN(x)	sine of x in radians
COS(x)	cosine of x in radians
TAN(x)	tangent of x in radians
MIN( <i>expression1,expression2</i> )	Returns the lesser value of the two expressions. The expressions must be of the same form as the primary Density <b>expression</b> , without the square brackets.
MAX( <i>expression1,expression2</i> )	Returns the maximum value of the two expressions. The expressions must be of the same form as the primary Density <b>expression</b> , without the square brackets.

In [Table 4-2](#), x is a numeric argument (including numeric variables and other numeric-valued expressions). There is no compile-time or runtime exception checking on the arguments of these functions, and unreasonable or undefined values may result from certain arguments.

The expression must not result in strictly negative values because such values cannot be checked by a **constraint**. Division by zero is defined *not* to satisfy any **constraint**.

Parentheses have the highest precedence for the calculation of numeric values and may be used for grouping of terms.

**Operators** — Unary operators (+, -, !, ~) require only one numerical argument and all such operators have the same precedence. The unary + and - operators are the usual positive and negative signs. The other unary operators do the following:

**! operator** — Returns 0 (or false) if its argument is non-zero and 1 (or true) if its argument is 0.

**~ operator** — Returns 0 (or false) if the argument is positive and 1 (or true) if the argument is non-positive.

[Table 4-3](#) shows some useful combinations of the  $\sim$  and  $!$  operators.

**Table 4-3. ! and  $\sim$  Operator Combinations**

x	$!(x)$	$!(x-1)$	$!!(x)$	$!!(x-1)$	$\sim(x)$	$\sim(x-1)$	$\sim\sim(x)$	$\sim\sim(x-1)$
3	0	0	1	1	0	0	1	1
2	0	0	1	1	0	0	1	1
1	0	1	1	0	0	1	1	0
0	1	0	0	1	1	1	0	0
-1	0	0	1	1	1	1	0	0
-2	0	0	1	1	1	1	0	0
-3	0	0	1	1	1	1	0	0

Here is a summary of interpretations of the  $!$  and  $\sim$  operators:

$!(x)$  detects whether something is absent (here, 0 means absent).

$!!(x)$  detects whether something is present (not 0 means present).

$!(x-r)$  detects the value r.

$!!(x-r)$  detects everything except r.

$\sim(x)$  detects whether something is absent (here,  $x \leq 0$  means absent).

$\sim\sim(x)$  detects whether something is present ( $x > 0$  means present).

$\sim(x-r)$  detects everything  $\leq r$ .

$\sim\sim(x-r)$  detects everything  $> r$ .

[Table 4-3](#) does not show these relations explicitly, but they can be verified from the table:

$\sim\sim(x) = \sim(x)$  and  $\sim\sim\sim(x) = \sim(x)$ .

The binary operators  $^*$ ,  $*$ ,  $/$ ,  $+$ ,  $-$  require two numerical arguments. The  $^*$  operator is the same as the C language `pow()` function, where  $x ^ y$  means x raised to the y power. The  $*$ ,  $/$ ,  $+$ , and  $-$  operators are multiplication, division, addition, and subtraction, respectively, and have the customary precedence. The  $^*$  operator has the same precedence as  $*$  and  $/$ .

The binary operators  $\parallel$  (OR) and  $\&\&$  (AND) are the same as the C language operators of the same type, except that they have the same precedence as binary  $+$  and  $-$ . They expect numeric-valued inputs. For example:

`AREA(via) && AREA(met)` returns 1 if both inputs are non-zero and 0 otherwise.

`AREA(via) || AREA(met)` returns 1 if either input is non-zero and 0 otherwise.

Here is an example of a Density operation using a *density\_expression*:

**DENSITY L1 L2 L3 [ ( AREA(L1) + AREA(L2) + AREA(L3) ) / AREA() ] ...**

This is equivalent to the default density calculation used without a *density\_expression*.

- **constraint**

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The **constraint** “ $< 0$ ” is not allowed.

When used without a *density\_expression*, data capture windows whose area ratio meets the **constraint** are output. In this case, if the ratio is intended as a percentage, the **constraint** should contain numbers between 0 and 1, where division by 100 has already occurred.

If a *density\_expression* is provided, then data capture windows whose expression values meet the **constraint** are output.

- **WINDOW {wxy | wx wy}**

An optional keyword set that specifies the dimensions of the data capture window within which the density check is to occur. The choices are:

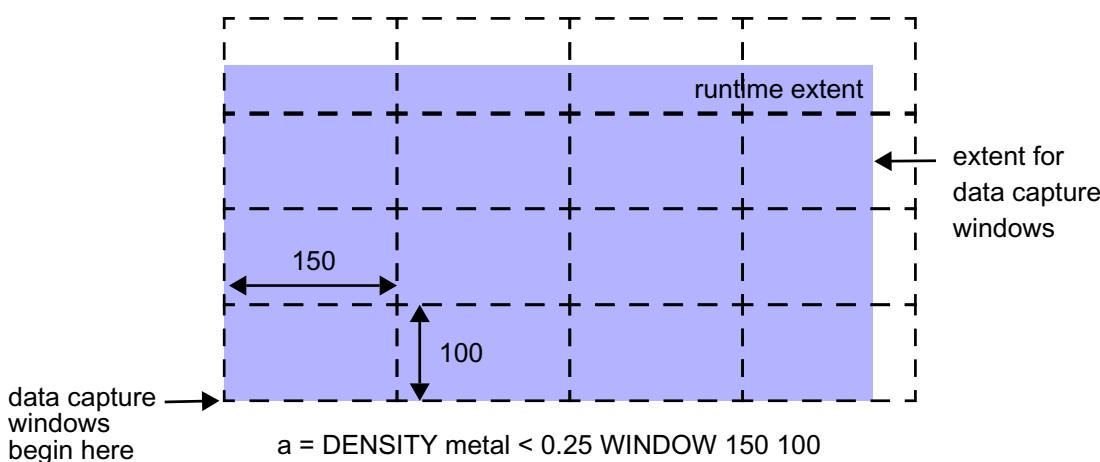
**WINDOW wxy** — Specifies a square window with a height and width of *wxy* user units. The parameter *wxy* must evaluate to a positive real number.

**WINDOW wx wy** — Specifies a rectangular window with a height of *wy* user units and a width of *wx* user units. The parameters *wx* and *wy* must evaluate to positive real numbers.

The windows are tiled across a data capture extent, which is determined by the keywords specified with the operation. Windows that meet the **constraint** are output. Windows that are written to the DRC Results Database are merged.

**Default behavior** — If you do not specify this keyword set in the statement, the default window size is either the extent of the layout database read in at runtime, or is defined by the **INSIDE OF** condition when one is specified. *Calibre does not read in database layers if they are not required for calculating outputs during the run.* If the **WINDOW** is larger than the boundary in the x-direction or y-direction, then the **WINDOW** is truncated to the boundary’s dimensions in the appropriate directions.

**Figure 4-18. Density WINDOW**



- STEP {sxy | sx sy}

An optional keyword set that is specified with WINDOW. Specifies the grid increment for which data capture windows are tiled within the data capture extent. The choices are:

STEP sxy — Specifies that the windows are tiled to the right and up in increments of sxy user units. The parameter sxy must evaluate to a positive real number.

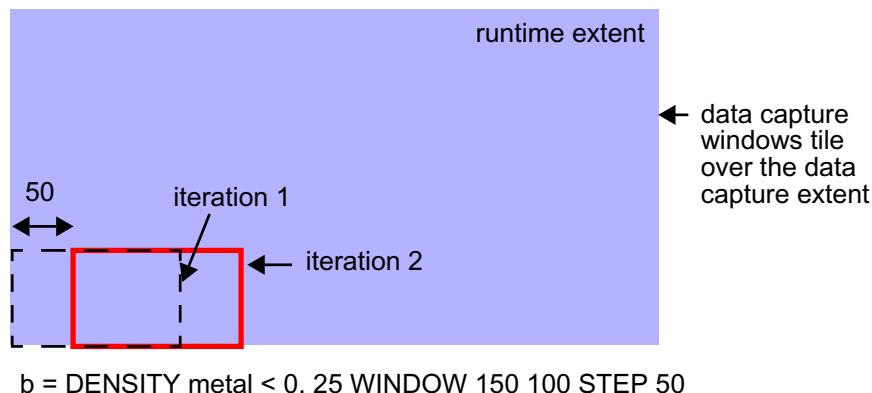
STEP sx sy — Specifies that the windows are tiled incrementally to the right sx user units and up sy user units. The parameters sx and sy must evaluate to a positive real numbers.

If you specify both STEP and WINDOW, then the values sxy, sx, and sy must evenly divide into the WINDOW values wxy, wx, and wy, respectively.

**Default behavior** — If you do not specify STEP in the statement, the default STEP size is defined by the WINDOW dimensions.

Note that STEP 3 -1 is interpreted as STEP 2, and STEP 3(-1) is a compiler error.

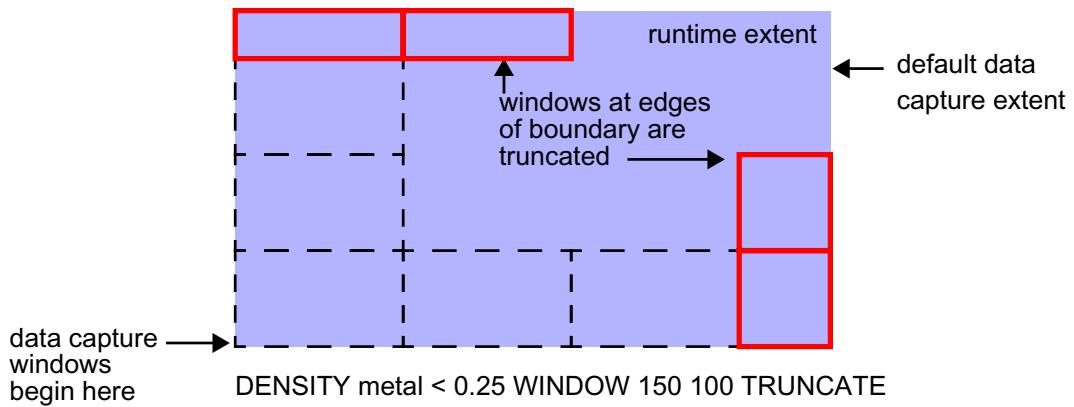
**Figure 4-19. Density WINDOW STEP**



The stepped data capture windows start in the lower-left corner of the data capture extent and initially are tiled in sxy or sx increments to the right. Upon arriving at the right edge of the data capture extent, the capture windows are tiled upward by the sxy or sy increment and are tiled again starting from the left edge of the data capture extent.

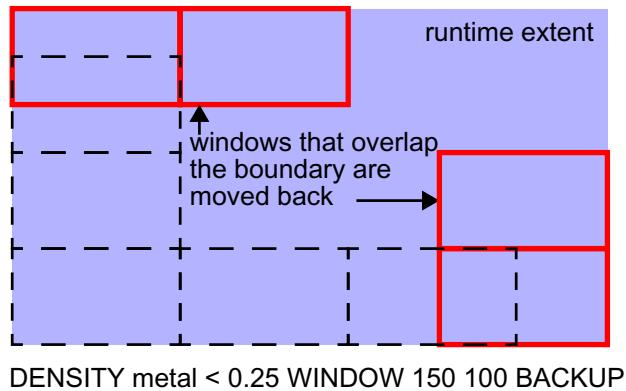
- TRUNCATE

Optional keyword that specifies a data capture window is truncated at the limits of the data capture extent. The density calculation in a truncated window is based upon the truncated dimensions of the window. This is the default behavior if you do not specify this keyword. The algorithms for this and related keywords are discussed later in the [Functional Details](#) section.

**Figure 4-20. Density WINDOW TRUNCATE**

- **BACKUP**

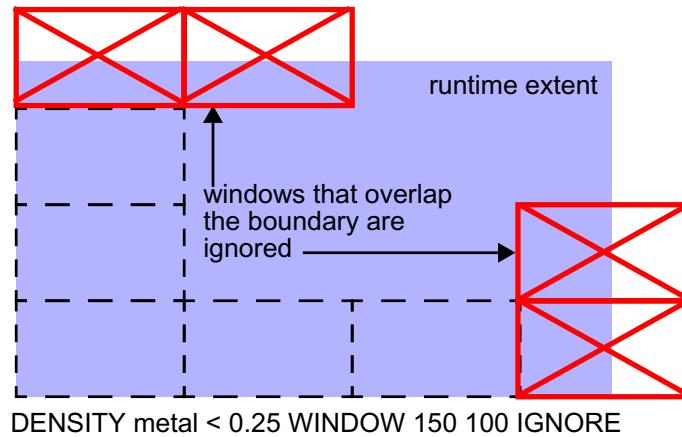
Optional keyword that specifies if a window overlaps the right-hand or top edges of the data capture extent, the window is shifted left or down until it is no longer overlapping the data capture extent. The Density calculation is then performed after the window has been shifted.

**Figure 4-21. Density WINDOW BACKUP**

- IGNORE

Optional keyword that specifies if a window overlaps the right-hand or top edges of the data capture extent, then the window is ignored and no data for that window location is output.

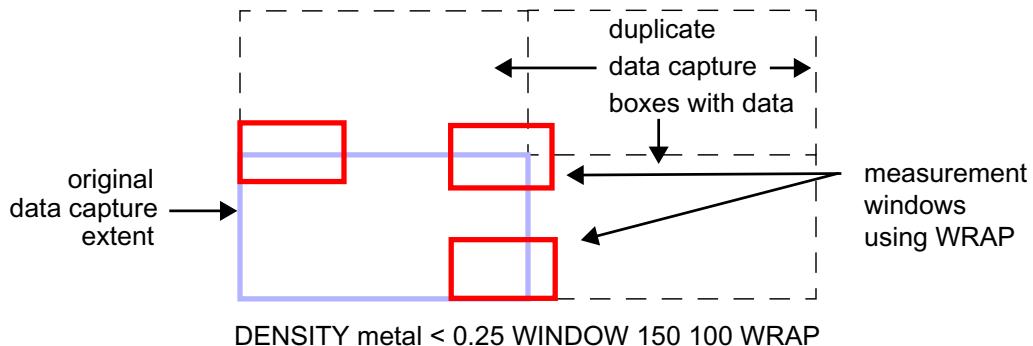
**Figure 4-22. Density WINDOW STEP IGNORE**



- WRAP

Optional keyword that specifies if a window overlaps the right-hand or top edges of the data capture extent, then the data capture extent and its data are duplicated and added to the right-hand side or top side of the original bounding box. The density measurement is then taken in the window that intersects the duplicated data capture extent regions.

**Figure 4-23. Density WINDOW STEP WRAP**



- INSIDE OF

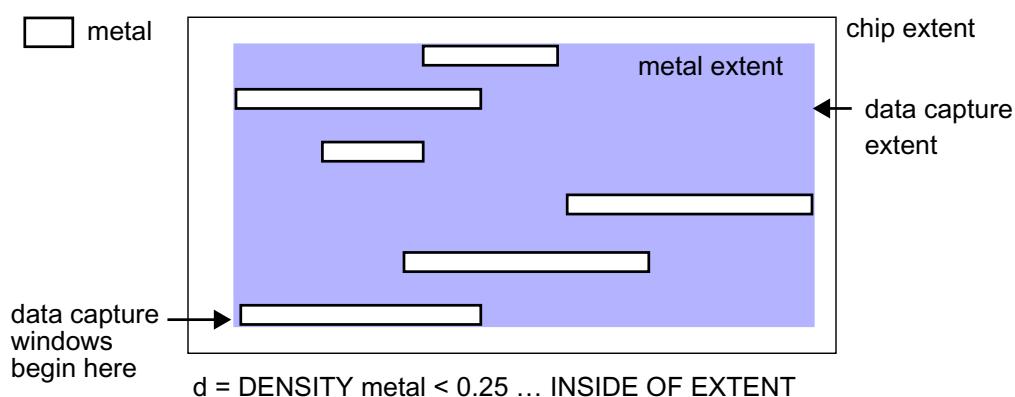
An optional keyword set that defines a data capture extent within which data capture windows are tiled.

**Default behavior** — If you do not specify any of these keyword sets in the operation, the default data capture extent is the database extent read in at run time. *Calibre does not read in database layers if they are not required for calculating outputs during the run.* Layers that are not read in do not contribute to the Calibre database extent.

The choices for this option are these:

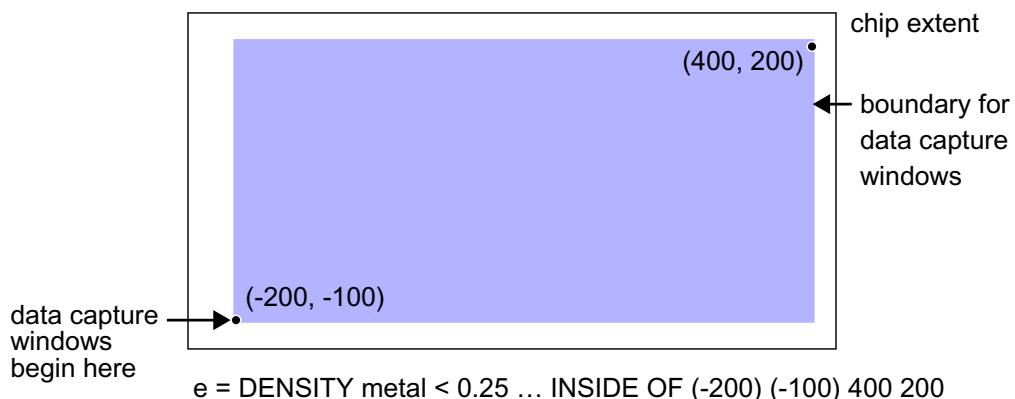
INSIDE OF EXTENT — Specifies that the data capture extent is the rectangular extent of the input *layer* parameters.

**Figure 4-24. Density INSIDE OF EXTENT**



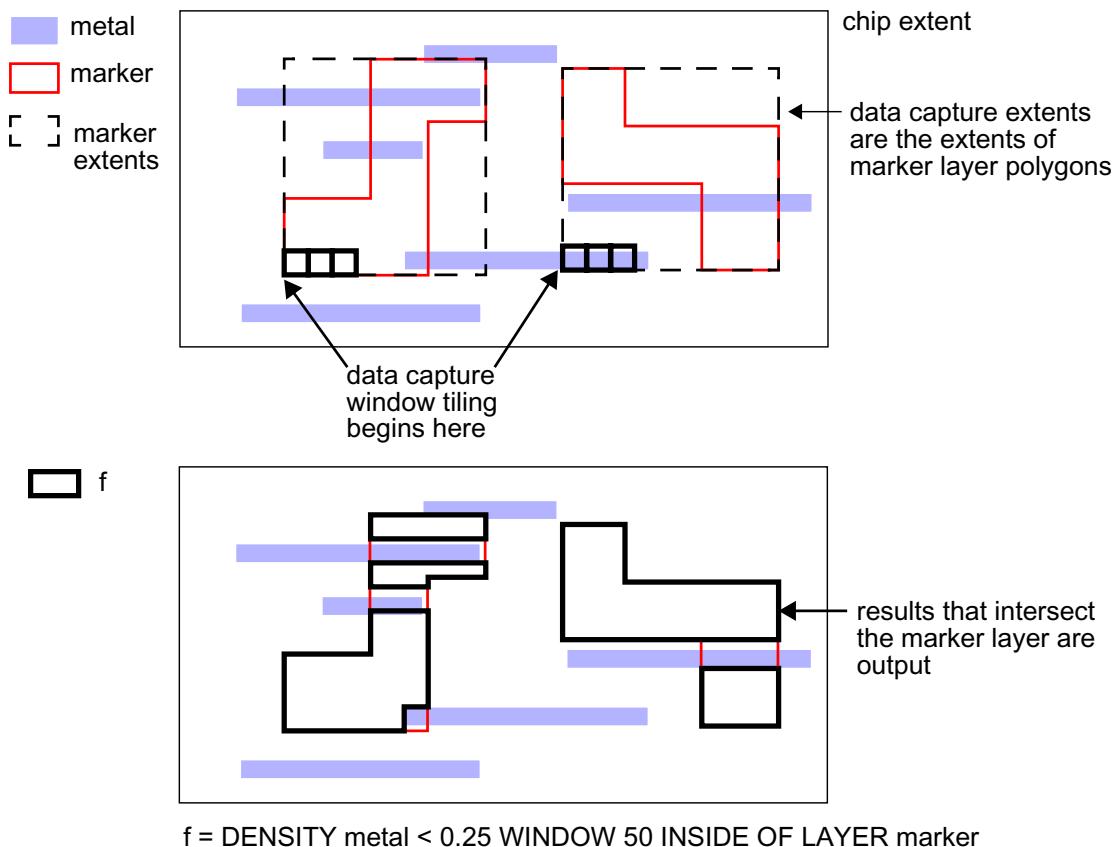
INSIDE OF  $x1\ y1\ x2\ y2$  — Specifies the lower-left and upper-right corners of a rectangular data capture extent, orthogonal to the database axes, in user units. If a coordinate is negative, it must be enclosed in parentheses ( ).

**Figure 4-25. Density INSIDE OF  $x1\ y1\ x2\ y2$**



**INSIDE OF LAYER *layerB*** — If no other keywords from the INSIDE OF LAYER family are used, specifies the output of data capture windows is coincident with polygons from *layerB*. The *layerB* parameter must be an original or derived polygon layer and may not be a *layerN* parameter. The data capture extents are the rectangular extents of polygons from *layerB*. [Figure 4-26](#) shows example results output.

**Figure 4-26. Density INSIDE OF LAYER**

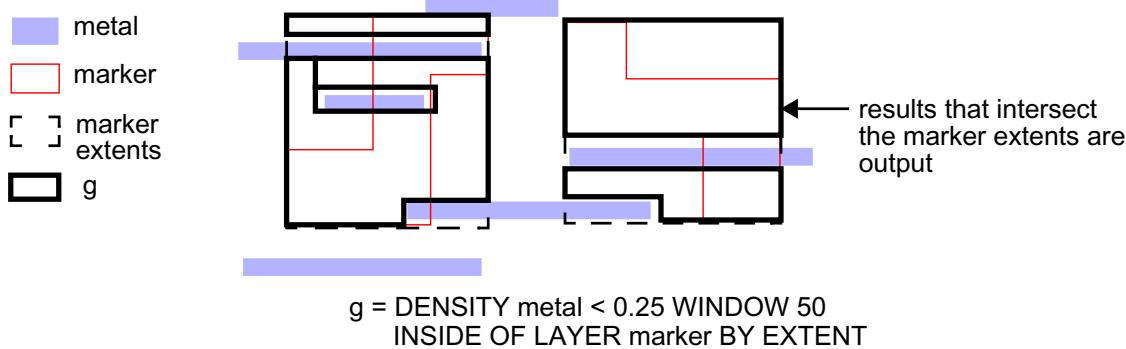


The following optional keyword sets can be specified with INSIDE OF LAYER and are mutually exclusive of one another. They modify the INSIDE OF LAYER results output and data capture extents. The exact algorithms for all of these options are in the [Functional Details](#) section.

The INSIDE OF LAYER option is costly from a performance standpoint and should generally be avoided.

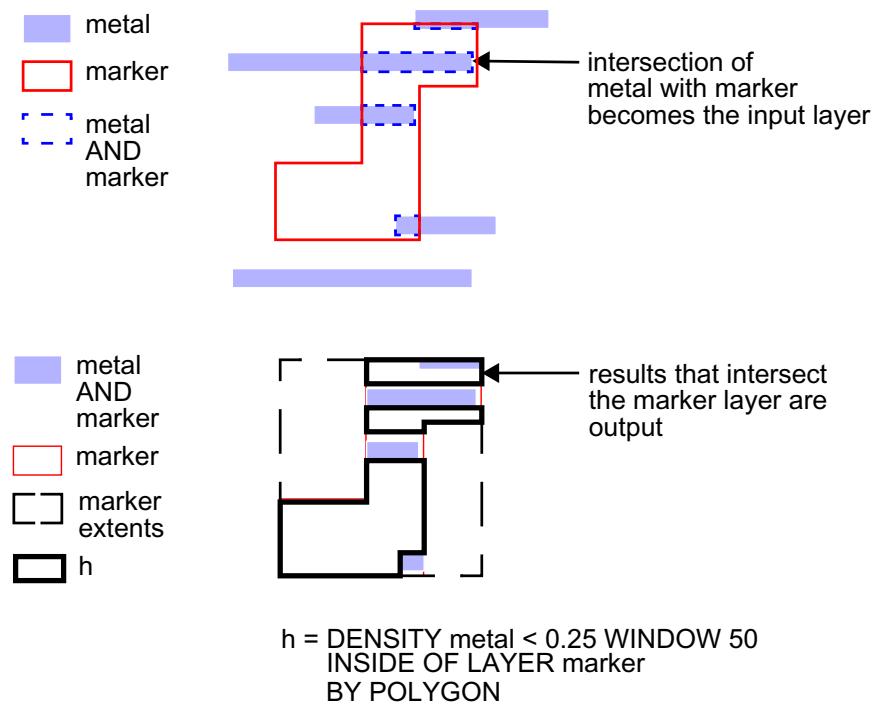
- o BY EXTENT [*size\_value*] — Specifies the output of data capture windows is coincident with the extents of polygons from *layerB*, not the polygons themselves. [Figure 4-27](#) shows example results output. Compare this to [Figure 4-26](#).

**Figure 4-27. Density INSIDE OF LAYER BY EXTENT**



The optional *size\_value* is a non-negative floating-point value in user units that causes the extents of the polygons from *layerB* to be expanded by the specified amount. The expanded extents are then used.

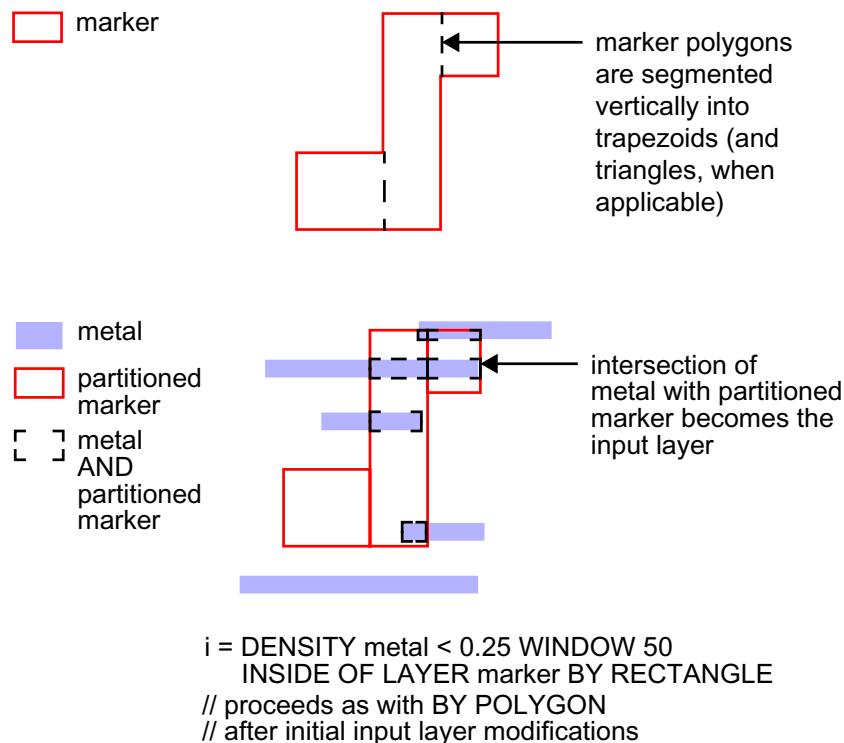
- o BY POLYGON — Specifies to modify the input polygons to be the intersections (Boolean AND) of polygons from the input *layer* parameters with polygons from *layerB*. These modified input polygons are then used for the density calculations. The output is coincident with polygons from *layerB*. May not be specified with COMBINE. [Figure 4-28](#) shows example results output.

**Figure 4-28. Density INSIDE OF LAYER BY POLYGON**

- BY RECTANGLE — Specifies to modify polygons on *layerB* by segmenting them vertically into trapezoids and triangles. Then the BY POLYGON algorithm is used with the segmented *layerB* polygons. May not be specified with COMBINE.

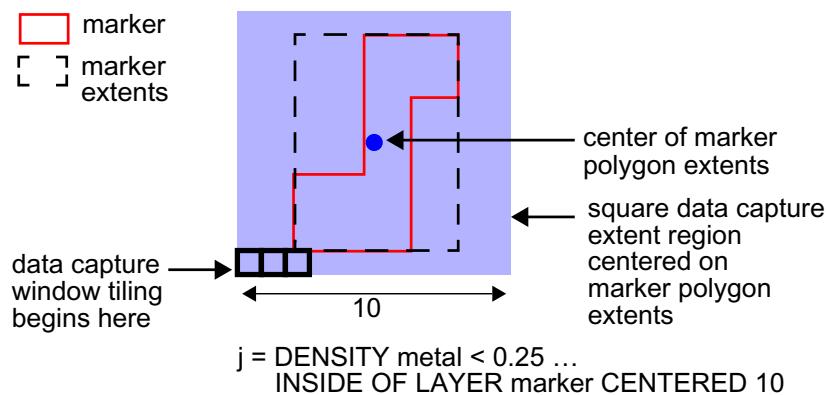
[Figure 4-29](#) shows the initial steps of the BY RECTANGLE algorithm. After these steps, the operation proceeds as with BY POLYGON.

**Figure 4-29. Density INSIDE OF LAYER BY RECTANGLE**



- CENTERED *value* — Specifies that squares centered inside the extents of *layerB* polygons are used for the data capture extents. The *value* is the side length of the square in user units.

**Figure 4-30. Density INSIDE OF LAYER CENTERED**



- GRADIENT *constraint* [RELATIVE | ABSOLUTE] [CORNER [*value*]]

Optional keyword set that specifies the gradient value a window must satisfy in order to be output. The gradient is a measure of density value change from one window to adjacent non-overlapping windows. In order for a data capture window to be output, it must satisfy both the usual Density *constraint* and the GRADIENT *constraint*.

The GRADIENT function G is defined as:

$$G(A, B) = || V_A | - | V_B || / \max(| V_A |, | V_B |) \text{ (RELATIVE equation)}$$

or

$$G(A, B) = || V_A | - | V_B || \text{ (ABSOLUTE equation)}$$

for windows A and B being compared, where  $V_A$  is the density value of window A and  $V_B$  is the density value of window B.

The following table shows the possible GRADIENT values for given values of  $V_A$  and  $V_B$ .

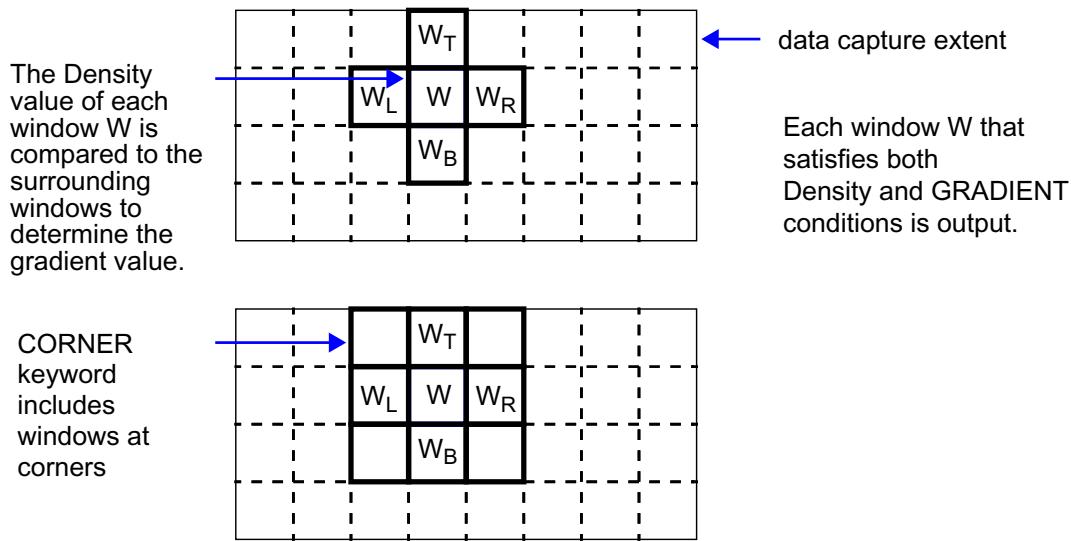
**Table 4-4. GRADIENT Values**

$ V_A $	$ V_B $	<b>G(A, B) RELATIVE</b>	<b>G(A, B) ABSOLUTE</b>
equals $ V_B $	equals $ V_A $	0	0
0	>0	1	$ V_B $
>0	0	1	$ V_A $
>0, not equal $ V_B $	>0, not equal $ V_A $	$  V_A  -  V_B   / \max( V_A ,  V_B )$	$  V_A  -  V_B  $

The gradient value of a window W is the *maximum* of this set:

$$\{G(W, W_L), G(W, W_R), G(W, W_B), G(W, W_T)\}$$

where  $W_L$  is the window immediately to the left of W,  $W_R$  is the window immediately to the right of W,  $W_T$  is the window immediately above W,  $W_B$  is the window immediately below W.

**Figure 4-31. GRADIENT Windows**

$G(W, W_L)$  is 0 if  $W$  is a leftmost window,  $G(W, W_R)$  is 0 if  $W$  is a rightmost window, and so forth.

Any window  $W$  that satisfies both the Density operation's primary **constraint** and the GRADIENT **constraint** is output.

The RELATIVE calculation is the GRADIENT default; the ABSOLUTE equation is used if the keyword ABSOLUTE is specified.

**CORNER** [*value*] — optional keyword and parameter set that specify the set of windows in the GRADIENT calculation is extended to include the four windows touching  $W$  on each corner, which may not exist at certain window locations. If *value* is specified, then each corner window is multiplied by the *value*. The value is a dimensionless, non-negative floating-point number. CORNER may only be specified with GRADIENT.

If GRADIENT and PRINT are both specified, then each line in the PRINT output file has an additional (sixth) numeric entry, which is the GRADIENT value of the window.

GRADIENT may not be specified with MAGNITUDE or COMBINE. If GRADIENT is specified with STEP, gradient values are calculated for abutting windows only, not for overlapping windows.

- **MAGNITUDE constraint** [RELATIVE | ABSOLUTE]

Optional keyword set that determines if there is at least one data capture window within the data capture extent that satisfies a specified gradient value (see the GRADIENT description for the definition), which is given by the MAGNITUDE **constraint**. In order for a data capture window to be output, it must satisfy both the usual Density **constraint** and the MAGNITUDE **constraint**.

Let  $A$  be a given data capture window. Let  $C(A)$  be the number (or magnitude count) of other windows,  $B$ , such that the gradient value, denoted  $G(A, B)$ , satisfies the MAGNITUDE **constraint**. Hence,  $C(A)$  is always a non-negative integer representing the

count of other windows that satisfy this condition. If  $C(A) > 0$ , then window A is output (this implies B will also be output because it will satisfy the MAGNITUDE condition with A). Note that the windows B whose ratio values do not satisfy the Density operation's primary **constraint** do not contribute to  $C(A)$ .

**Figure 4-32. MAGNITUDE Windows**

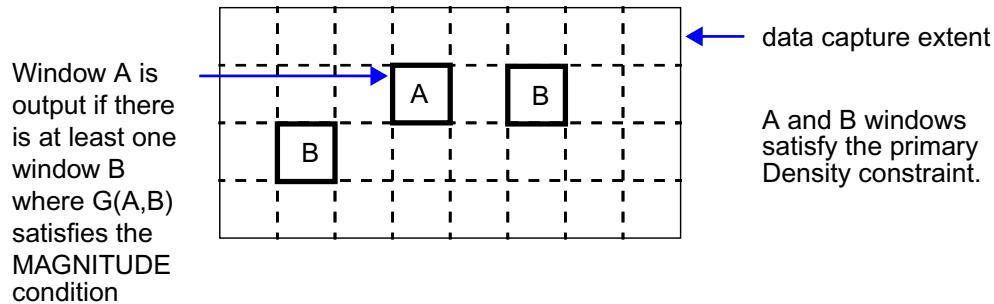


Table 4-4 shows the possible gradients in the RELATIVE and ABSOLUTE modes for all combinations of  $V_A$  and  $V_B$  values, which are the Density values of windows A and B respectively. The RELATIVE calculation is the default; the ABSOLUTE equation is used if the keyword ABSOLUTE is specified.

Note that, whereas the calculation used by the GRADIENT keyword compares only windows adjacent to A, the MAGNITUDE keyword performs the gradient calculation for *all* data capture windows other than A within the INSIDE OF data capture extent, which satisfy the Density operation's primary condition.

Given the preceding definitions, any window A that generates a  $G(A, B)$  value that satisfies the MAGNITUDE *constraint* is output.

When you specify MAGNITUDE, the PRINT [ONLY] file and the RDB [ONLY] file have different formats from the default Density output for these files. See [PRINT Output](#) or [RDB Output](#) for details on file formats.

The MAGNITUDE option processing time increases proportionally to the square of the number of windows. The number of data capture windows should not be large when using MAGNITUDE.

MAGNITUDE may not be specified with GRADIENT or COMBINE.

- **CENTERS** *value*

Optional keyword and positive floating-point number in user units that cause the geometric output of Density to be squares of *value*  $\times$  *value* dimensions located at the centers of rectangles that would be output by default. May not be specified with COMBINE.

- PRINT [ONLY] *filename*

An optional keyword set that specifies that all processed window data are printed to the desired files. The choices are these:

PRINT *filename* — Specifies that the tool prints the density window data in ASCII form to the specified *filename* in addition to the usual results output.

PRINT ONLY *filename* — Specifies that the tool prints the density window data in ASCII form to the specified *filename* only. The PRINT ONLY parameter leaves the results database layer empty. This saves operation time if you are not interested in the geometric result. This parameter is only supported in Calibre nmDRC applications. All other applications issue a warning and do not generate the printed output; however, they do generate the standard output of the operation.

File names are case-sensitive. Subdirectories in a *filename* are created as required. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in chapter 2, “[Key Concepts](#)”.

- RDB [ONLY] *filename* [MAG *value*]

An optional keyword set that instructs the tool to create an ASCII results database (RDB) with geometry and detailed statistics, having the given *filename*. File names are case-sensitive. Subdirectories in a pathname are created as required.

RDB *filename* — Specifies RDB output to the *filename*. This database is *in addition to* the usual nmDRC Results Database (see [DRC Results Database](#)) and can be loaded into Calibre RVE or Pyxis Layout.

RDB ONLY *filename* — Specifies results are sent to the RDB only. No geometric data is sent to the usual DRC Results Database. Any results sent to the RDB in this mode *are not reported* in either the run transcript or the DRC Summary Report.

MAG *value* — Specifies the results sent to the RDB *filename* are magnified by the specified *value*. The *value* must be a positive floating point number, which is dimensionless. When specified, the MAG keyword overrides a [DRC Magnify Density](#) statement.

RDB is for use in nmDRC-related applications only. If used in nmLVS or PEX applications, a warning is issued and the Density operation continues as if RDB were not specified.

- COMBINE *constraint* [RELATIVE | ABSOLUTE]

An optional keyword set that controls the format of the RDB file. It offers a method for combining windows in the RDB file whose density ratio values differ by the amount specified in the *constraint*.

The COMBINE behavior is *in addition to* the standard criterion that the ratio value of the window must satisfy the Density operation’s primary *constraint*. The COMBINE keyword set must be specified with either the RDB or RDB ONLY keywords. See the [COMBINE Output](#) section for details of the resulting RDB structure.

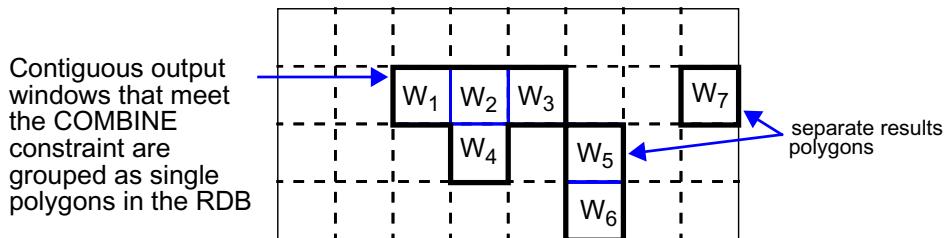
## Density

When you specify the COMBINE keyword set, the following algorithm is applied.

Let  $\{ W_1, \dots, W_n \}$  be any set of output windows (that is, windows whose ratio value satisfies the primary Density constraint) such that the following are true:

- Each window  $W_i$  in the set that touches or overlaps some window  $W_j$  in the set (that is, the set of windows is contiguous).
- For each  $W_i$  in the set, when the difference between its ratio value and that of all other windows  $W_k$  in the set satisfies the COMBINE *constraint*, group such  $W_i$  windows in the RDB as a single polygon.

**Figure 4-33. COMBINE Output**



The density value difference function  $D(A, B)$  between any two windows in the set  $\{ W_1, \dots, W_n \}$  must satisfy the COMBINE *constraint* for the set of windows to be combined. The function  $D$  is defined as follows:

$$D(A, B) = || V_A | - | V_B || / \max(| V_A |, | V_B |) \text{ (RELATIVE equation)}$$

or

$$D(A, B) = || V_A | - | V_B || \text{ (ABSOLUTE equation)}$$

for windows  $W_A$  and  $W_B$ , where  $V_A$  is the density value of  $W_A$  and  $V_B$  is the density value of  $W_B$ .

$D(A, B)$  is defined to be 0 in RELATIVE mode (the default) if both  $V_A$  and  $V_B$  are 0.

RELATIVE is the default mode if neither RELATIVE nor ABSOLUTE is specified.

The COMBINE keyword may not be specified with GRADIENT, MAGNITUDE, CENTERS, BY POLYGON, or BY RECTANGLE.

## Description

Generates a derived polygon layer by measuring the density of its input layer(s) over a user-specified grid within a user-specified rectangular data capture extent.

If no *density\_expression* is supplied, Density outputs rectangles of a specified WINDOW size whose density ratio meets the *constraint*. That is, if the ratio of the total area of input layers in a data capture window to the area of the window itself satisfies the *constraint*, then the window is output.

A ***constraint*** ratio that is a percentage, such as 20%, should be entered as 0.20, not 20. The value 20 is permitted, but such a value will only generate results when used with a proper *density\_expression*.

Density operations are performed concurrently if all parameters, except ***constraint*** and PRINT [ONLY], are the same.

## Functional Details

The exact algorithm follows:

In a bounding box with corners (BX1,BY1), (BX2,BY2), which corresponds to the data capture extent for the operation, is first established. If INSIDE OF  $x1\ y1\ x2\ y2$  is specified (see [Figure 4-25](#)), this data capture extent is given by the four numeric parameters, which must designate a valid rectangle orthogonal to the database axes. Numeric arguments are in user units. If INSIDE OF EXTENT is specified (see [Figure 4-24](#)), the data capture extent is the extent of the input layers. If neither one is specified, then the data capture extent is the database extent read in at runtime (this may not properly contain the input layer due to oversizing with Size and so forth, if applicable) that is read in at runtime. Remember that Calibre only reads in the layers that are actually required to produce output for the run, which may not be the entire set of database layers.

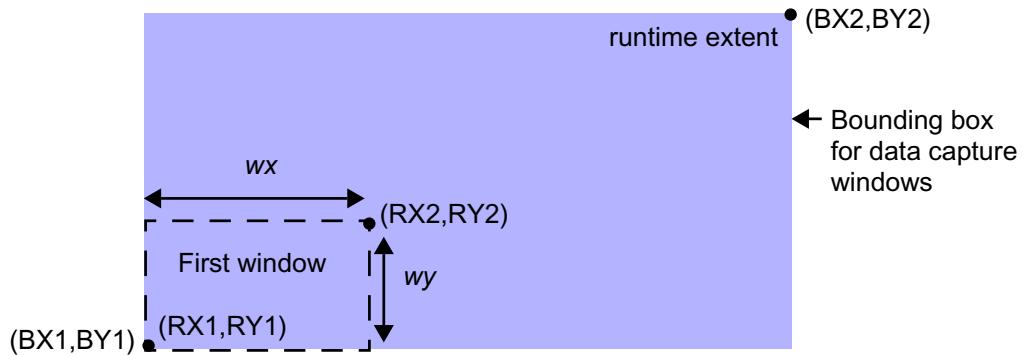
The WINDOW parameter specifies the minimum rectangle size in which the density check is to occur (see [Figure 4-18](#)). This rectangle size is  $wx$  by  $wy$ . If only one value is specified,  $wxy$ , the rectangle is assumed to be a square where  $wx = wy$ . If WINDOW is not specified, it defaults to the size of the bounding box (BX1,BY1), (BX2,BY2) defined previously. Arguments  $wx$  and  $wy$  must be positive numbers and are in user units.

The STEP parameter specifies the step size to tile the data capture window at each iteration (see [Figure 4-19](#)). This step size is  $sx$  by  $sy$  if both are specified. If only one value is specified,  $sxy$ , then  $sx = sy$ . STEP has these conditions:

- STEP parameters must all be positive numbers and are interpreted in user units.
- STEP must be specified with WINDOW.
- If STEP is not specified, then  $sx$  and  $sy$  default to the WINDOW dimensions  $wx$  and  $wy$ , respectively.
- If both WINDOW and STEP are specified, then  $sx$  and  $sy$  must be less than or equal to  $wx$  and  $wy$ , and evenly divide  $wx$  and  $wy$ , respectively.
- If the WINDOW parameter exceeds the data capture extent size in the x-direction, that is, if  $wx > (BX2 - BX1)$ , then both  $wx$  and  $sx$  are set to  $BX2 - BX1$ . Similarly for the y-direction.

Given the bounding box determined by (BX1,BY1), (BX2,BY2), the WINDOW size  $wx \times wy$ , and the STEP size  $sx \times sy$  defined previously, the operation executes conceptually as follows:

1. Initialize the iteration by placing a rectangle R of size  $wx \times wy$  in the lower-left corner of the bounding box (BX1,BY1), (BX2,BY2). At each step, designate R's lower-left and upper-right coordinates by (RX1,RY1), (RX2,RY2). Go to Step 2.

**Figure 4-34. Bounding Box and WINDOW Coordinates**

2. If no *density\_expression* is provided, the ratio of the total input layer area in the rectangle R to the area of R itself is compared to the *constraint*. If the ratio meets the *constraint*, then R is output. Otherwise the *density\_expression* is computed for the layer data in R and compared to the *constraint*. If the expression meets the constraint, then R is output. If  $RX2 > BX2$  or  $RY2 > BY2$ , then R is truncated by the bounding box for use in this calculation (see [Figure 4-20](#)). Go to Step 3.
3. If  $RX2 \geq BX2$  then move R back to the left side of the bounding box and go to Step 4. Otherwise, move R to the right by  $sx$  and go to Step 2.
4. If  $RY2 \geq BY2$  then quit. Otherwise, move R up by  $sy$  and go to Step 2. When complete, all output rectangles are merged.

TRUNCATE, BACKUP, IGNORE, and WRAP are optional, mutually exclusive keywords.

If TRUNCATE is specified, then there is no change in the previous algorithm. TRUNCATE explicitly states the default algorithm is to be used.

If BACKUP is specified, then Step 2 is altered so that if  $RX2 > BX2$ , then the rectangle R is not truncated, but rather is backed up so that  $RX2$  is equal to  $BX2$ . Similarly for the y-direction. The area of the backed-up rectangle, and that of the input layer within the backed-up rectangle, are then used in the calculation. See [Figure 4-21](#).

If IGNORE is specified, then Step 2 is altered so that if  $RX2 > BX2$ , then the rectangle R is completely ignored and no output is generated from this rectangle, similarly for the y-direction. In addition, if the WINDOW size initially exceeds the data capture extent size in either the x- or y-direction, then there is no output. See [Figure 4-22](#).

If WRAP is specified (see [Figure 4-23](#)), then the initial algorithm steps are modified as follows (throughout this discussion, the term density can be replaced with *density\_expression* with no loss of generality):

1. Unchanged from default algorithm.
2. If the ratio of the layer's area in the rectangle R to the area of R itself meets the *constraint*, then R is output. If  $RX2 > BX2$  or  $RY2 > BY2$ , then the density is calculated as if the entire bounding box and its data were duplicated at all of the three locations  $(BX2,BY1)$ ,  $(BX2,BY2)$ , and  $(BX1,BY1)$ .

3. If  $RX2 = BX2$  or  $RX1 \geq BX2$ , then move R back to the left side of the bounding box and go to Step 4. Otherwise, move R to the right by  $sx$  and go to Step 2.
4. If  $RX2 = BY2$  or  $RY1 \geq BY2$  then quit. Otherwise, move R up by  $sy$  and go to Step 2.

The INSIDE OF LAYER *layerB* [BY EXTENT | BY POLYGON | BY RECTANGLE | CENTERED *value*] keyword set limits the density computation to the polygonal areas of *layerB* (which must be an original or derived polygon layer). Depending on the optional keywords, the input layers and data capture extent are also changed. This option is expensive from a performance standpoint and should generally be avoided.

Assume the operation is:

**DENSITY L1 L2 ... Ln INSIDE OF LAYER B**

If INSIDE OF LAYER is specified with no optional keywords (see [Figure 4-26](#)), then for each polygon P in layer B:

1. Compute the rectangular extent  $(X1, Y1) \rightarrow (X2, Y2)$  of P.
2. Perform the operations as if INSIDE OF X1 Y1 X2 Y2 was specified instead of INSIDE OF LAYER. This step determines the RDB and PRINT output, if these keywords are specified.
3. Perform a Boolean AND of geometric output from Step 2 with P.
4. Accumulate RDB and PRINT output from Step 2 (if any) and geometric output from Step 3 into the total output from the operation.

If BY EXTENT [*size\_value*] is specified (see [Figure 4-27](#)), then for each polygon P in layer B:

1. Compute the rectangular extent  $(X1, Y1) \rightarrow (X2, Y2)$  of P.
2. Perform the operation as if INSIDE OF X1 Y1 X2 Y2 was specified instead of INSIDE OF LAYER.
3. Accumulate geometric, RDB, and PRINT output from Step 2 into the total output of the Density operation.

If *size\_value* is specified, then the extent of P calculated in Step 1 is oversized by *size\_value*, and the coordinates of the expanded rectangle are used.

Note that the extents of non-rectangular polygons on layer B are used, not the polygons themselves.

If BY POLYGON is specified (see [Figure 4-28](#)), then for each input layer  $L_n$  and each polygon P in layer B:

1. Set layer  $A_1 = L_1 \text{ AND } P, A_2 = L_2 \text{ AND } P, \dots, A_n = L_n \text{ AND } P$  (these are Boolean AND operations).
2. Compute the rectangular extent  $(X1, Y1) \rightarrow (X2, Y2)$  of P.

3. Perform the operation as if the input layers were  $A_1, A_2, \dots, A_n$  and as if INSIDE OF X1 Y1 X2 Y2 is specified instead of INSIDE OF LAYER. This step determines the RDB and PRINT output, if these keywords are specified.
4. Take geometric output from Step 3 and perform a Boolean AND with P. Filter out RDB and PRINT output rectangles that do not have area intersection with P.
5. Accumulate geometric, RDB, and PRINT output from Step 4 into the total output from the operation. The RDB output consists of P itself if WINDOW is not specified.

The BY POLYGON keyword is potentially useful in situations where *layerB* has serpentine polygons and the output of data capture windows near these polygons requires the handling this keyword set provides.

If BY RECTANGLE is specified (see [Figure 4-29](#)), then for each polygon P in layer B:

1. Segment into trapezoids and triangles by cutting P vertically at each unique x coordinate value of a vertex of P.
2. For each trapezoid or triangle T, perform the BY POLYGON algorithm using T instead of P.

If CENTERED *value* is specified (see [Figure 4-30](#)), then for each polygon P in layer B:

1. Compute the center point (CX,CY) of the rectangular extent of P.
2. Set X1 = CX - V, Y1 = CY - V, X2 = CX + V, and Y2 = CY + V where V is *value* / 2.
3. Perform the operation as if INSIDE OF X1 Y1 X2 Y2 was specified instead of INSIDE OF LAYER.
4. Accumulate geometric, RDB, and PRINT output from Step 3 into the total output from the operation.

The CENTERED option can be useful in situations where there are markers (polygons or text objects) that you want to serve as the centers of bounding boxes in which the Density check is performed.

## **PRINT Output**

The PRINT or PRINT ONLY parameters produce an ASCII file containing the coordinates of the WINDOW rectangle, followed by the density ratio. The coordinates, in order, are lower-left x, lower-left y, upper-right x, upper-right y. For example:

```
1646 -1557.75 1647 -1556.75 0.5
1646 -1556.75 1647 -1555.75 0.675
1646 -1555.75 1647 -1554.75 0.554
1646 -1554.75 1647 -1553.75 0.5
1646 -1553.75 1647 -1552.75 0.267
1646 -1552.75 1647 -1551.75 0.438
```

The file is sorted by the lower-left coordinate (x then y) of the rectangles. The output print file can get very large. You should select the **constraint** value and the WINDOW and STEP values to minimize the number of generated rectangles. The rectangles are not generally merged in the output print file as in the result layer.

The GRADIENT option causes the GRADIENT value to be appended to each line of the file.

The MAGNITUDE option causes the magnitude count (a dimensionless positive integer defined in the Parameters section discussion of MAGNITUDE) to be appended to each line representing a WINDOW rectangle. Following the line for  $W_A$  is a listing of all windows  $W_B$ , one per line (a total of magnitude count lines), such that the gradient value  $G(A, B)$  satisfies the MAGNITUDE keyword's constraint. Each line consists of the coordinates of  $W_B$ , then the ratio value of  $W_B$ , followed by the gradient, as follows:

- $W_A$  file entry: <x1><y1><x2><y2><ratio><magnitude count>
- $W_B$  file entry: <x1><y1><x2><y2><ratio><gradient>

For example:

```
(DENSITY L WINDOW 100 STEP 50 > 0.5 < 1 MAGNITUDE > .495 ABSOLUTE PRINT
file)
...
8713.6 -2162.2 8813.6 -2062.2 0.998 0.497168      <--- WB
8713.6 3637.8 8813.6 3737.8 0.998 0.497168      <--- WB
8863.6 37.8 8963.6 137.8 0.995936 0.495104      <--- WB
-8836.4 -3212.2 -8736.4 -3112.2 0.50432 6      <--- WA
-6786.4 4837.8 -6686.4 4937.8 0.999488 0.495168      <--- WB
-1986.4 4037.8 -1886.4 4137.8 0.99992 0.4956      <--- WB
-1986.4 4087.8 -1886.4 4187.8 0.99992 0.4956      <--- WB
-636.4 3537.8 -536.4 3637.8 0.9996 0.49528      <--- WB
-636.4 3737.8 -536.4 3837.8 0.9996 0.49528      <--- WB
8313.6 -562.2 8413.6 -462.2 0.99944 0.49512      <--- WB
-8836.4 -512.2 -8736.4 -412.2 0.500832 84      <--- WA
-8336.4 137.8 -8236.4 237.8 0.997756 0.496924      <--- WB
-7586.4 -3012.2 -7486.4 -2912.2 0.996348 0.495516 <--- WB
...
```

Note that each ( $W_A, W_B$ ) pair will also be in the print file elsewhere as ( $W_B, W_A$ ).

## RDB Output

RDB output is somewhat similar to PRINT output. The RDB option creates an ASCII nmDRC results database file *in addition to* the standard [DRC Results Database](#). The RDB contains a single nmDRC rule check corresponding to the Density operation.

The first output line includes the top-cell name database precision as specified by the [DRC Results Database Precision](#) statement, with 1000 being the default. The name of the rule check is the name of the layer created by the operation. The first line of checktext is the text of the Density operation itself (excluding RDB parameters). If the Density operation has output to the DRC Results Database and has check text comments (@), then those comments are shown also. The number of DRC results in the rule check is the number of (unmerged) output windows, each window being an individual DRC result.

For example:

```
TOP 1000
k::<1>
14621 14621 1 Jul 21 10:03:17 2003
k::<1> = DENSITY M1 PG >=.25 WINDOW 105 GRADIENT >=0
[AREA(M1)+AREA(PG)/AREA( )]
p 1 4
DV 546
DG 0.8839
DA 11025
DA M1 546
DA PG 0
-8986400 -4272200
-8881400 -4272200
-8881400 -4167200
-8986400 -4167200
p 2 4
DV 756
DG 0.8445
DA 11025
DA M1 756
DA PGM 0
-8986400 -3852200
-8881400 -3852200
-8881400 -3747200
-8986400 -3747200
...
```

Notice the presence of properties DV, DG, and DA:

DV (Density value) — density value of the window (to eight significant figures).

DG (Density gradient) — gradient value of the window if GRADIENT is specified.

DA (Density area) — may either be followed by a numeric value, or by a *layer* name and then a numeric value. With no layer name, it is the area of the window and is always attached. Otherwise, a DA property is attached for each input *layer* and is the layer name followed by its area in the window.

Properties can be mapped to colors for display in RVE. They can also be sent to a histogram in RVE for analysis. See “[Creating Histograms and Colormaps](#)” in the *Calibre Interactive and Calibre RVE User’s Manual* for details.

If MAGNITUDE is specified, then an additional property, DMC (Density magnitude count), is present (a dimensionless positive integer whose definition appears in the Parameters section discussion of MAGNITUDE). In addition, the results, each representing a window  $W_B$  such that the gradient value satisfies the MAGNITUDE keyword’s *constraint*, are present between the result for  $W_A$  and the result for the next output window. The number of these additional results is the magnitude count, and the total result count in the nmDRC rule check corresponding to the Density operation is adjusted accordingly.

For example:

```

TOP 1000
XYZ
6543 6543 1 Jan 23 13:48:43 2007
XYZ = DENSITY L > 0.5 < 1 WINDOW 100 STEP 50 MAGNITUDE > 0.495 ABSOLUTE
p 1 4      <--- WA
DV 0.500832
DA 10000
DA L 5008.32
DMC 3      <--- DMC property, indicating there are 3 WB windows
-8836400 -4112200
-8736400 -4112200
-8736400 -4012200
-8836400 -4012200
e 2 4      <--- WB
DM 0.500832 0.997756 0.496924
-8836400 -4112200 -8736400 -4112200
-8836400 -4012200 -8836400 -4112200
-8336400 137800 -8236400 137800
-8336400 237800 -8336400 137800
e 3 4      <--- WB
DM 0.500832 0.996348 0.495516
-8836400 -4112200 -8736400 -4112200
-8836400 -4012200 -8836400 -4112200
-7586400 -3012200 -7486400 -3012200
-7586400 -2912200 -7586400 -3012200
e 4 4      <--- WB
DM 0.500832 0.996928 0.496096
-8836400 -4112200 -8736400 -4112200
-8836400 -4012200 -8836400 -4112200
-7136400 4837800 -7036400 4837800
-7136400 4937800 -7136400 4837800
...

```

The format of each nmDRC result for W<sub>B</sub> entry allows a viewing environment to show (W<sub>A</sub>,W<sub>B</sub>) as a single entity. Each W<sub>B</sub> entry in the database is a 4-edge cluster: the bottom edge of W<sub>A</sub>, then the left edge of W<sub>A</sub>, then the bottom edge of W<sub>B</sub>, then the left edge of W<sub>B</sub>. In addition, each nmDRC result W<sub>B</sub> has a DM (Density magnitude) property that consists of three floating-point values: the density value of W<sub>A</sub>, then the density value of W<sub>B</sub>, and then the gradient value.

Note that each (W<sub>A</sub>,W<sub>B</sub>) pair will also be in the RDB file elsewhere as (W<sub>B</sub>,W<sub>A</sub>).

If you specify RDB ONLY, then the Density check results only go to the RDB and not to the usual nmDRC results database. The results are not reported in the DRC Summary Report, if specified.

RDB results databases may be used in common among multiple Density RDB operations in the rule file. The first open of an RDB file in a nmDRC run truncates the file and writes the header line consisting of the top-cell name and precision. Subsequent file opens are in append mode. If an RDB file cannot be opened, then a warning is issued and the operation continues as if RDB were not specified. The RDB name is checked for collisions with [DRC Results Database](#) or [DRC Check Map](#) database names. A conflict generates a compiler error. The names of RDBs generated by other layer operations are not checked.

**Note**

If the RDB *filename* is quoted in one Density operation but not another, the later of the two operations in the rule file overwrites the RDB of the earlier operation. String constants in the rule file are handled differently from other strings.

---

The geometric results in the RDB can be magnified by using either the MAG keyword in the operation itself, or through the [DRC Magnify Density](#) specification statement. If both are used, the MAG keyword takes precedence.

## COMBINE Output

If you specify the COMBINE keyword set, then the set of windows {  $W_1, \dots, W_n$  } are not output into the RDB database as individual windows. Rather, they are merged into a single polygon that is output with two additional auxiliary results. For example:

```
...
p 7 11
DAVG 0.00558239      <--- average density of combined windows
DWC 176                <--- number of combined windows
-5124500 -804500
-5024500 -804500
-5024500 -704500
-4974500 -704500
-4974500 -654500
-4874500 -654500
-4874500 -604500
-4824500 -604500
-4824500 -304500
-4874500 -304500
-4874500 -204500
p 8 4
DMIN 0                 <--- minimum window value
-5124500 -804500
-5024500 -804500
-5024500 -704500
-5124500 -704500
p 9 4
DMAX 0.9825            <--- maximum window value
-5074500 795500
-4974500 795500
-4974500 895500
-5074500 895500
...

```

The first polygon represents the merged set of windows {  $W_1, \dots, W_n$  }. The properties that appear are defined as follows:

DAVG — average density value of all the windows in the set.

DWC — total number of windows in the set (that is,  $n$  from  $W_n$ ). The polygon coordinates are then followed by two auxiliary results.

DMIN — minimum density value from a window in the set {  $W_1, \dots, W_n$  }. The vertices of the corresponding window are provided.

DMAX — maximum density value from a window in the set {  $W_1, \dots, W_n$  }. The vertices of the corresponding window are provided.

## Examples

Examples of Density usage can also be found under “[Density Best Practices](#)” in the *Calibre Solutions for Physical Verification* manual.

### Example 1

The following example checks the specification: The density of metal2 in every  $50 \times 50$  area of the layout must exceed 25%:

```
met2_check {
    @ The density of metal2 in every 50 x 50 area of the layout must
    @ exceed 25%
    DENSITY metal2 < 0.25 WINDOW 50
}
```

### Example 2

The following example specifies a 2 user unit step size because “3 -1” is viewed as the arithmetic operation 3 minus 1:

```
DENSITY metal2 < 0.25 WINDOW 10.0 STEP 3 -1
```

whereas, the following example results in a compilation error due to the negative y-value:

```
DENSITY metal2 < 0.25 WINDOW 10.0 STEP 3 (-1) // error
```

### Example 3

Metal density in any  $100 \times 100$  window, stepped  $50 \times 50$ , must exceed 0.25. However, if there is poly present in the window, then there is no requirement on metal density. Solution:

```
// division by 0 in the expression does not meet the constraint
density_rule_a { DENSITY metal poly <= 0.25 WINDOW 100 STEP 50
    [ AREA(metal) / ( !AREA(poly) * AREA() ) ]
}
```

### Example 4

Metal density in any  $100 \times 100$  window, stepped  $50 \times 50$ , must exceed 0.25. However, if there is poly present in the window, then the area of the poly must first be subtracted from the window area. Results are sent to a separate RDB database. Solution:

```
density_rule_b{ DENSITY metal poly <= 0.25 WINDOW 100 STEP 50
    [ AREA(metal) / ( AREA() - AREA(poly) ) ] RDB density_rdb
}
```

## Density

---

### Example 5

Metal density in any  $100 \times 100$  window, stepped  $50 \times 50$ , must exceed 0.25. However, if there is poly present in the window, then the area of the poly must be first subtracted from the window area. In addition, the metal area must exclude any intersection of metal and poly. Results are sent to a separate RDB database. Solution:

```
density_rule_c{
    x = metal NOT poly
    DENSITY x poly <= 0.25 WINDOW 100 STEP 50
    [ AREA(X) / ( AREA() - AREA(poly) ) ] RDB density_rdb
}
```

### Example 6

Derive a layer based upon the density of a layer L within certain regions of the design that are marked by text objects labeled “xyz”, and the regions are  $100 \times 100$  squares centered on the text objects. Solution:

```
x = EXPAND TEXT "xyz" BY .01
y = DENSITY L <= 0.25 WINDOW 100 INSIDE OF LAYER x CENTERED 100
```

# Density Convolve

Layer operation

```
DENSITY CONVOLVE input_layer [ '['density_expr ']'] constraint [WINDOW wxy]
[INSIDE OF EXTENT | INSIDE OF x1 y1 x2 y2 | INSIDE OF LAYER layer2]
[TRUNCATE | WRAP] [PRINT [ONLY] filename]
FILE {parameter_block | '}' 
  {{SCALE cn GAUSS sn [CONVOLUTION_GRID cgn] [NORMALIZATION type]...}... |
   {SCALE cn FRACTAL {nn | filenamen} [RMIN rminn] [RMAX rmaxn]
    [CONVOLUTION_GRID cgn] [PARTIAL r1n r2n] [NORMALIZATION type]
    [skip]...}}
  '}' 
  [spatialPolynomial [a0] [a1x] [a2y] [a3xy] [a4x2y] [a5xy2] ...]
  [upsample_interpolation [linear | none]]
  [vboasis_path file_path [-noCheck]]
  [vboasis_precision_multiplier {n[/ d] | AUTO}]
  [vboasis_layout_magnify n [/ d]]
  [vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
  [<SVRFSTART>
    svrf_statements
  <SVRFEND> ]
  [svrf_layer_name {layer_name | {layer_name oasis_layer_number
    [oasis_layer_datatype]...}}]
  [density_precision [double | float]]
  [dose_ring dose delta_x_um delta_y_um]
  [blade_shadow blade_x_um blade_y_um]
  [RDB [ONLY] filename [MAG value]]
  [globalMapFile {binary | ascii} file_path]
  [section_size sz]
  }
```

## Summary

The Density Convolve layer operation allows you to create a representation of your input layer that shows density values after applying convolution algorithms. Convolution is a mathematical procedure that produces a weighted moving average of two functions, in this case it evaluates the density value of a density window by taking into consideration the values of those windows surrounding it. Therefore the density values are “spread out” to nearby density windows.

## Parameters

- *input\_layer*

The name of an original or derived polygon layer. This is the layer containing the data to which Calibre applies the convolution functionality.

This differs from [Density](#) in that you can only specify a single layer.

- ‘[’ *density\_expr* ‘]’

An optional expression enclosed in square brackets ( [ ] ) that allows customized control over the density computations. The expression may contain numbers, numeric variables, binary operators ( \*, / , +, -), unary operators (+, -, !, ~), and AREA functions of the input layer of the form, where the parentheses ( ) are required:

```
AREA ( input_layer )
```

The expression must not result in strictly negative values because they cannot be checked by a constraint. The AREA function returns values based upon user units of length squared. AREA() returns the area of the data capture window.

For example, to return the value “1 - density” to the convolution operation you would specify:

```
[ 1 - AREA(layer_name) / AREA( ) ]
```

- ***constraint***

A required argument that is a constraint listed in Table 2-2.

If the density value, after convolution, of a window meets this constraint, the window is output to the derived layer.

If a *density\_expression* is provided, the result of the expression is tested against the constraint.

If the ratio is intended as a percent, it should be a number between 0 and 1, where division by 100 has occurred. For example, if you want the constraint to be 20%, you would specify “.20” not “20”.

You cannot specify the constraint “< 0”.

- **WINDOW *wxy***

An optional keyword set that specifies the size of the window within which the tool calculates the density and applies the convolution algorithm.

Specifies a square window with a height and width of *wxy* user units. The string *wxy* must be a positive real number. If you do not specify this optional keyword set, the default window size is defined by the INSIDE OF boundary, resulting in a single window.

To prevent any WINDOWS from overlapping, the tool STEPs the window by the same value (*wxy*).

Consult your process engineer for the best size of the density window. Typically, this would be about two orders of magnitude larger than the nominal feature size.

- **INSIDE OF EXTENT | INSIDE OF *x1 y1 x2 y2* | INSIDE OF LAYER *layer2***

An optional keyword set that defines a rectangular boundary within which the tool creates the density grid. The choices are:

INSIDE OF EXTENT — Specifies that the rectangular boundary is the extent of *input\_layer*.

**INSIDE OF**  $x1\ y1\ x2\ y2$  — Specifies the lower-left and upper-right corners of the rectangular boundary in user units. You must write the coordinates relative to the origin of the Calibre database and enclose any negative coordinates in parentheses ( ), for example:

INSIDE OF (-123) 123 (-12) 456

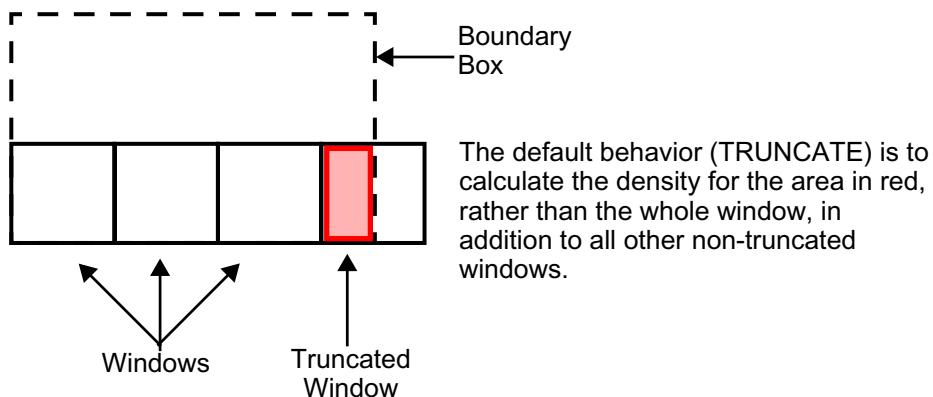
**INSIDE OF LAYER**  $layer2$  — Specifies that the rectangular boundary is the extent of  $layer2$ . This is different behavior from the Density SVRF statement and  $layer2$  cannot be the same layer as  $layer1$ .

- **TRUNCATE** | WRAP

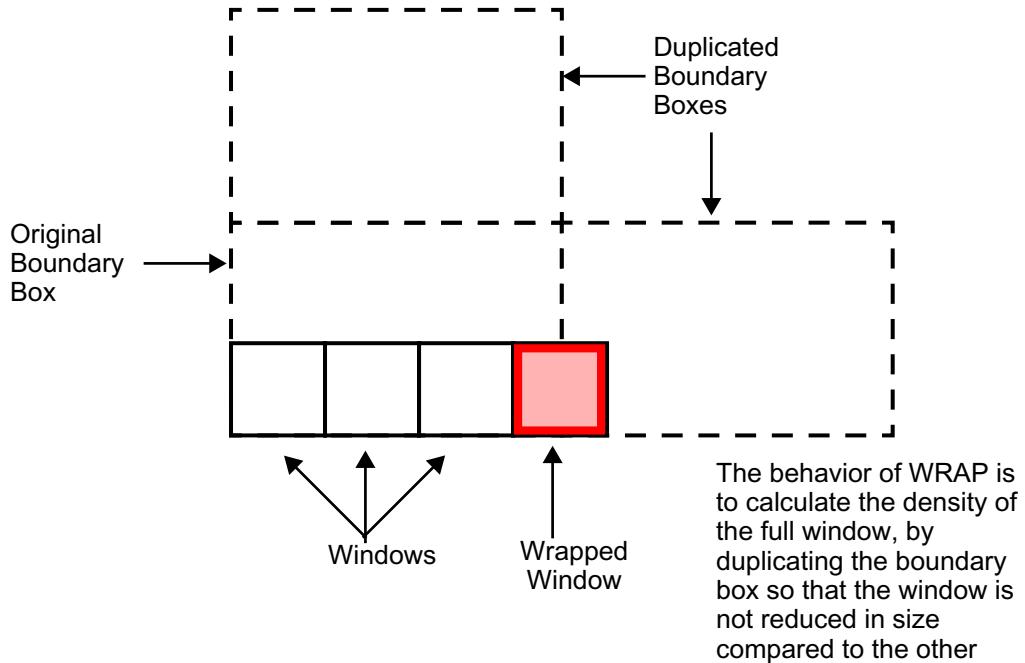
Specifies how the tool calculates the density of a window that overlaps the boundary box of the *input\_layer*.

**TRUNCATE** — An optional keyword that instructs the tool to compute the density of the overlapping window by truncating the size of the window so that it does not extend beyond the boundary box. This is the default behavior.

**Figure 4-35. TRUNCATE Keyword**



**WRAP** — An optional keyword that instructs the tool to compute the density of the overlapping window by duplicating and adding the boundary box, and its data, to the right-hand or top side of the main boundary box. The density measurement is then taken in the window that intersects the duplicated boundaries.

**Figure 4-36. WRAP Keyword**

- **PRINT [ONLY] *filename***

An optional keyword set that instructs the tool to write data about each window to the desired location. The choices are:

**PRINT *filename***—Writes the window data to the specified *filename* and the result layer.

**PRINT ONLY *filename*** — Writes the window data only to the specified *filename* and leaves the result layer empty. This saves operation time if you are not interested in the geometric result.

The *filename* parameter can contain environment variables.

The file is a list of all the windows in the following format:

x1 y1 x2 y2 value  
where,

- x1 y1 x2 y2 — reports the position of the window
- value — reports the density value, after convolution

- **FILE *parameter\_block***

A keyword/argument pair specifying that the density convolution parameters are contained in a reusable parameter block defined using the [Litho File SVRF statement](#).

- **FILE** '[' ... ']'

A keyword/argument set specifying that the density convolution parameters are contained inline within the square brackets ( [ ] ). Square brackets must surround all operation arguments appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines *strictly between* the left and right brackets (that is, cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **SCALE**  $c_n$

A required keyword/argument pair specifying a scaling factor. This factor is associated with each two-dimensional Gaussian kernel.

- **GAUSS**  $s_n$

A required keyword/argument pair specifying a Gaussian sigma value. This value represents the sigmas for each two-dimensional Gaussian kernel.

- **FRACTAL** { $n_n$  | *filename<sub>n</sub>*}

A fractal value, representing the power of the two dimensional Fractal kernel that will be convolved with the density values. Note that Fractal kernels may not be combined with Gaussian kernels, but you may have multiple Fractal kernels. The value of  $n$  must be positive. Alternatively, a filename can be specified that contains a text based fractal kernel that will be read in and used for the convolution. The text file must contain a series of samples that strictly adheres to the following format:

```
// this is a comment
# this is also a comment
// All white space is ignored.
// This is a list of sample values of radius in microns followed by
// the fractal value.
// Only white space is allowed between the radius and fractal value.
<r1> <f1>
<r2> <f2>
...
<rn> <fn>
```

- **RMIN**  $rmin_n$

Specifies a Fractal minimum radius value in microns, representing the cutoff value for which the Fractal kernel will be set to zero if  $0 < r < rmax_n$ . The default value is 1.0 um.

- **RMAX**  $rmax_n$

Specifies of Fractal maximum radius value in microns, representing the cutoff value for which the Fractal kernel will be set to zero if  $r > rmax_n$ . The default value is 90% of energy in fractal kernel.

- CONVOLUTION\_GRID  $cg_n$

An optional keyword/argument pair specifying a convolution grid ( $cg_n$ ) for each two-dimensional Gaussian kernel.

You can use this setting to speed up the convolution of the density grid at the expense of precision.

The default value is the value assigned to GAUSS ( $s$ ) divided by 4.

- PARTIAL  $r1_n r2_n$

Specifies a partial fractal kernel, where only the portion  $r1_n < r_n < r2_n$  of the original fractal kernel is convolved at the given convolution grid. In this manner, the entire original range of the fractal kernel must be built up by using partial kernels. This allows you to customize the convolution grid accuracy and runtime trade-offs associated with very large fractal kernels. It is highly recommended to use “upsample\_interpolation linear” when using partial fractal kernels to avoid “stair-stepping” when combining the partial fractal kernels.

- NORMALIZATION { area | none }

Optional normalization flag for each kernel. The “area” option normalizes the convolution kernel to have an area of 1.0, which is then multiplied by the scale factor, while “none” performs no normalization on the convolution kernel but is multiplied by the scale factor. The default setting is area.

- skip

This secondary keyword is only used by fractal kernels. It specifies that this particular kernel is supposed to be “skipped” (not performed). This is useful when you would like a particular partial range of the kernel to be handled by the EUV flare simulator.

- spatialPolynomial [ $a0$ ] [ $a1x$ ] [ $a2y$ ] [ $a3xy$ ] [ $a4x2y$ ] [ $a5xy2$ ] ...

Defines up to a 4th order polynomial that is evaluated on the user units of the layout that is added to the convolution output. Coefficients can be negative or positive, while the order term must place ‘x’ before ‘y’. Only non-zero coefficients need to be specified.

- upsample\_interpolation [linear | none]

Specifies that Calibre use linear interpolation on upsampling to reduce “stair-stepping” from mismatched density and convolution grids. By default, upsampling is performed without linear interpolation.

- vboasis\_path *filepath* [-noCheck]

A file path to the optional direct input OASIS file. If this parameter is specified, input geometrical data for the density computation is taken from the specified OASIS file instead of the input hierarchical layer. All data in the file is merged to form the input to the density computation. The specified extent must match that of the OASIS file. Several restrictions and caveats apply to the use of this alternate input method:

- Limited compile-time extent checking is possible since the extent of the OASIS file is not known until it is parsed entirely. The extent is thoroughly checked at run time.

- For distributed processing (MTflex) with 32-bit slaves, the maximum allowed data size of any cell in the OASIS file is 2GB, although the practical limit will be determined by the slave memory. Files generated by conversion from any fracture format will always have sufficiently small cells.
- Computation mode. Any VB:OASIS file may be used for section mode. Hierarchical mode processing requires that the input VB:OASIS file only contain trapezoids and “orthogonal, unitary” placement transforms; the trapezoids are preserved when possible and the input is essentially translated to the output format. Hybrid mode processing may not be used with this keyword. It requires the Calibre hierarchical database.
- Oasis file indices. Section mode will always attempt to reuse an existing index file for the input VB:OASIS file. Hierarchical mode will always rebuild the index in order to check that the file contains only trapezoids. Any index constructed by the fracture command, whether in hierarchical or section mode, will not contain viewer components, and so may not be suitable for use in the viewer.
- Any file specified by this keyword must have only one layer.

While any valid VB:OASIS file (considering the restrictions) can be used as an argument to this keyword, use only OASIS.VSB files (a standardized subset of VB:OASIS) or the outputs of the MDPview format-to-oasis converters. These files have hierarchies that can be effectively used as spatial indices. Using VB:OASIS files not constrained by this trait will likely result in unacceptably poor fracture performance.

The -noCheck option can be specified in circumstances when checks on the input VBOASIS layout file are not possible because it has not been generated. This would typically happen when the file is expected to be generated by another SVRF command and thus is not available during parsing for checking purpose. When using -noCheck, extra care should be given to specifying the file path and associated precision values to avoid errors that will be encountered much later.

Several side effects occur when input is taken from a VB:OASIS file. The HDB precision is temporarily replaced by the precision in the VB:OASIS file for the duration of the fracture command. Also, the default HDB extent may no longer be a meaningful way of specifying the fracture window, since there is no relationship between the HDB and input OASIS file.

- vboasis\_precision\_multiplier {*n* [ / *d*] | AUTO}

When direct input is specified using the vboasis\_path keyword, this keyword will multiply the precision of the oasis file before operating on it. The precision will be multiplied by the ratio created by the numerator (*n*) and optional denominator (*d*) integer input. The default for *d* is 1 if otherwise unspecified.

If the AUTO is supplied instead of a ratio, Calibre will attempt to infer the correct ratio but will fail if it is not a rational number, or if the combination of inferred numerator and denominator would cause arithmetic overflow. This keyword requires the use of embedded SVRF.

- `vboasis_layout_magnify n [ / d ]`

Aligns the OASIS-direct-input coordinate system with the HDB coordinate system by adjusting its scale. The scale is specified as the numerator and denominator of a rational number.

This keyword is primarily used to apply embedded SVRF at the scale of the mask, but after all inverse transforms have been applied to align the MDP file inputs with the HDB coordinate system. In this case one would scale up the HDB coordinate system using LAYOUT MAGNIFY, and also use this keyword to apply the same scale to the direct-input database.

- `vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]`

The VB:OASIS file specified using vboasis\_path may contain cells that have large amounts of data and have a large extent. These large cells degrade performance in section mode processing. To overcome this issue, cells can be partitioned into bins.

The size of the cells that should be binned and the size of each bin (*bin\_size*) is determined automatically by default when vboasis\_injection parameter is specified. The user can control the bin size by providing a *size\_multiplier* using the option -size\_factor *size\_multiplier* (the default value is 1.0).

Alternately the *bin\_size* can be explicitly specified using the option -size *bin\_size* in microns. Any cell whose extent has width or height more the *bin\_size* is considered for binning.

The vboasis\_injection option creates temporary data which may be reused if it is preserved using the -preserve option (the default is 0).

The temporary data is read or written in a directory named *file\_path.lcb* where *file\_path* is argument specified to vboasis\_path. The location of this directory is same as that of *file\_path*. This behavior can be changed by specifying a colon-separated list of directories using the UNIX environment variable MDPIIndexSearchPath. The directories are searched in the order they are listed for reading or writing the temporary directory.

- <SVRFSTART>

A keyword indicating the beginning of the embedded SVRF statement group. Further information on using embedded SVRF can be found in the *MDP User's Manual*.

- *svrf\_statements*

Embedded SVRF statements must obey the same rules as SVRF with the additional constraints:

- Lines cannot be longer than 1023 characters (Calibre truncates any SVRF line to 1023 characters if you exceed this value).
- One output rule check with a single polygon output is allowed.
- No left- or right-brackets are allowed.

You must specify a single rule check within the embedded SVRF statement group. The output of this rule check is what Calibre uses as the input to the FRACTURE operation, and must only be polygon data, not edge or error data. Refer to the [Calibre Verification User's Manual](#) for more information about rule checks.

You cannot specify left- or right-brackets within the embedded SVRF statement group. Therefore, any occurrence of brackets must be replaced as follows:

Left bracket ( [ )	<LB>
Right bracket ( ] )	<RB>

The precision of the embedded SVRF statements must match that of the Calibre rule deck containing the FRACTURE statement.

- <SVRFEND>

A keyword indicating the end of the embedded SVRF statement group.

---

**Note**



The left and right angle brackets (<>) for <SVRFSTART>, <SVRFEND>, <LB>, and <RB> are literal and required. SVRFSTART and SVRFEND must also always be capitalized.

---

- svrf\_layer\_name {*layer\_name* | {*layer\_name oasis\_layer\_number* [*oasis\_layer\_datatype*]}}...

A keyword that specifies the names (*layer\_name*), numbers (*oasis\_layer\_number*), and optional data types (*oasis\_layer\_datatype*) of the embedded SVRF layers corresponding to VB:OASIS geometry. For a single layer, the first form is used. For multiple layers, each name and corresponding layer number in the OASIS file must be specified. When VB:OASIS input is used with HDB input, the precisions of the HDB and VB:OASIS file must match.

The parameters enclosed in curly braces ({*layer\_name* | {*layer\_name oasis\_layer\_number* [*oasis\_layer\_datatype*]}}) can be repeated 0 or more times.

- density\_precision [double | float]

This keyword instructs Calibre to internally store the density values with double or floating point precision, respectively. Significant memory savings will occur with float. The default setting is double.

- dose\_ring *dose delta\_x\_um delta\_y\_um*

Optional keyword that adds a ring of constant additional dose around the border of the convolution area given by points that are within *delta\_x\_um* of the left and right edges, and *delta\_y\_um* of the top and bottom edge. The additional dose is ADDITIVE in the corners of the simulation, meaning *dose* will be added twice for any points that satisfy both *delta\_x\_um* and *delta\_y\_um*. This is useful when modeling Optical Density or Black Border Reflections. The default is all values of zero. The *delta* values are in microns, and the *dose* value is in intensity.

- `blade_shadow blade_x_um blade_y_um`

Optional keyword that models the partial shadowing of the wafer by masking blades in the litho tool. This performs much like the `dose_ring` keyword, except it provides a linear roll-off of the dose from the value `dose` in the `dose_ring` command to zero at the end of the blade shadowing. Hence, this keyword requires that `dose_ring` set a non-zero `dose` value. In this manner, values within  $\delta_x \text{ um} < x < (\delta_x \text{ um} + \text{blade}_x \text{ um})$  of the right and left borders of the region, will experience a linear roll-off from `dose` to zero. Values within  $\delta_y \text{ um} < y < (\delta_y \text{ um} + \text{blade}_y \text{ um})$  of the top and bottom borders of the region will experience a linear roll-off from `dose` to zero. This effect is additive for any points satisfying both x and y conditions. The `blade` values are in microns.

- `RDB [ONLY] filename [MAG value]`

An optional keyword set that instructs the tool to create an ASCII results database (RDB) with geometry and detailed statistics, having the given `filename`. File names are case-sensitive. Subdirectories in a pathname are created as required.

`RDB filename` — Specifies RDB output to the `filename`. This database is *in addition to* the usual nmDRC Results Database and can be loaded into Calibre RVE or Pyxis Layout.

`RDB ONLY filename` — Specifies results are sent to the RDB only. No geometric data is sent to the usual DRC Results Database. Any results sent to the RDB in this mode *are not reported* in either the run transcript or the DRC Summary Report.

`MAG value` — Specifies the results sent to the RDB `filename` are magnified by the specified `value`. The `value` must be a positive floating point number, which is dimensionless.

- `globalMapFile {binary | ascii} file_path`

Specifies a user-defined grid of points that exactly matches the grid of the density function. The points in the grid are added to the convolution result, similar to the `spatialPolynomial` keyword. You must specify either binary or ASCII file format. The ease of use of an ascii file can be weighed against the smaller file size and faster read in of a binary file. Both options are available. Within the global map file, two integers must be written that specify the number of points in the X-direction and the number of points in the Y-direction. These integers are followed by a matrix of floating-point numbers that specify the global map. The global map must be written in column major order (columns rasterized first).

- `section_size sz`

Sets the desired size of a square section in user units. This can only be used with direct data entry, and overrides the default chosen settings.

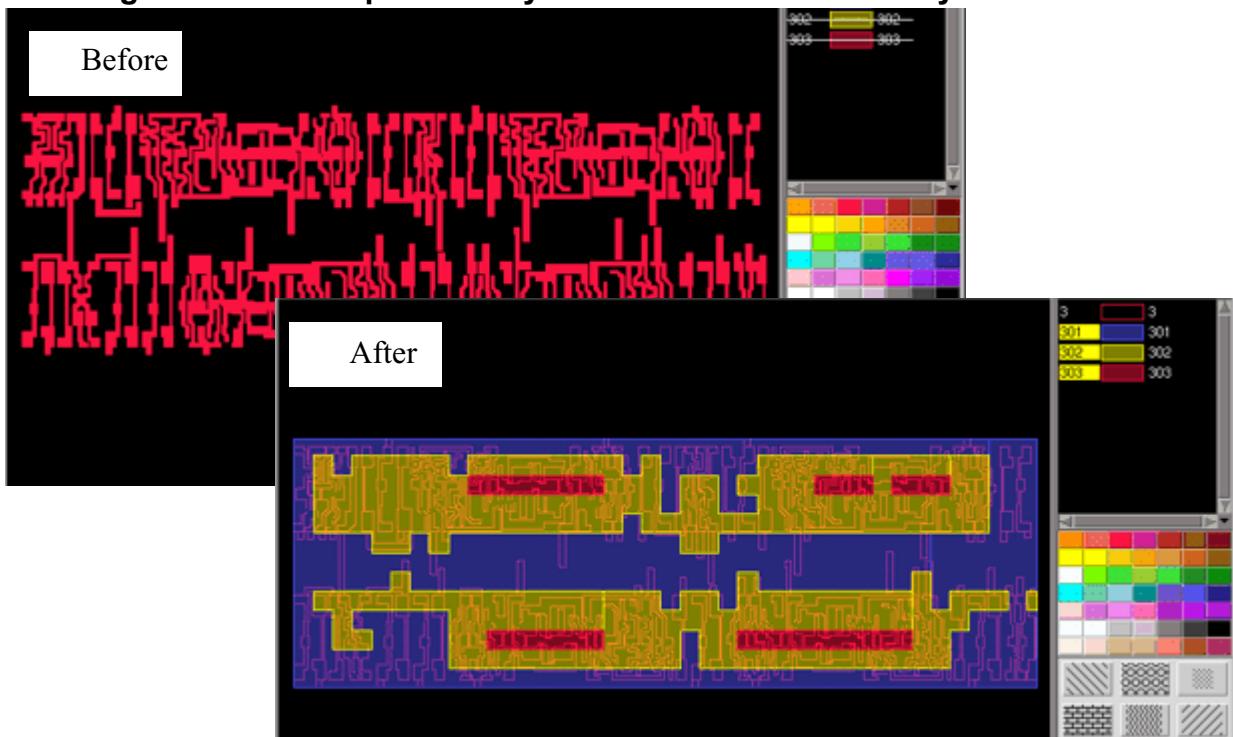
## Description

The Density Convolve operation allows you to create a representation of your input layer that shows density values after applying convolution algorithms. Convolution is a mathematical procedure that produces a weighted moving average of two functions, in this case it evaluates

the density value of a density window by taking into consideration the values of those windows surrounding it. Therefore, the density values are “spread out” to nearby density windows.

[Figure 4-37](#) shows two images. The first displays the input layer and the second displays the layers showing the density convolution.

**Figure 4-37. Example of a Layer Before and After Density Convolution**



Density Convolve behaves similarly to Density in that it creates a density grid across the extent of your data. One major difference in the creation of the grid is that no overlapping sections are allowed, therefore the step size is equivalent to the window size. Calibre calculates a density value for each window based on a specified *density\_expression*.

The tool then applies a convolution algorithm to the density value for each window. You specify the settings for the algorithm, also referred to as a two-dimensional Gaussian kernel, within the FILE argument.

The output of Density Convolve is a layer containing a merged representation of all the windows where the density value, after convolution, satisfies the specified *constraint*.

Typically you will write multiple Density Convolve statements to show a range of density values, making use of the Litho File statement. For example:

```
LITHO FILE convolve_parameters [
    SCALE 0.2 GAUSS 1
    SCALE 0.8 GAUSS 48
]

convolve_out_1 = DENSITY CONVOLVE 3 < 0.3 TRUNCATE WINDOW 25
FILE convolve_parameters
```

```
convolve_out_2 = DENSITY CONVOLVE 3 >= 0.3 < 0.4 TRUNCATE WINDOW 25
FILE convolve_parameters

convolve_out_3 = DENSITY CONVOLVE 3 >= 0.4 TRUNCATE WINDOW 25
FILE convolve_parameters
```

Calibre executes Density Convolve statements concurrently when they use the same settings for the *input\_layer*, WINDOW, INSIDE OF, and TRUNCATE/WRAP keywords. This allows Calibre to compute the initial density values only once for concurrent Density Convolve statements, saving processing time.

## Examples

### Example 1

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "../data/ring.oas"
LAYOUT PRIMARY "ring"

PRECISION 1000
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "convolve_ring_output.oas" OASIS PSEUDO
DRC SUMMARY REPORT report.convolve_ring

LAYER orig3 3

LITHO FILE convolve_parameters [
    SCALE 0.2 GAUSS 1
    SCALE 0.8 GAUSS 48
]

convolve_out_1 = DENSITY CONVOLVE 3 < 0.3 TRUNCATE WINDOW 25 FILE
convolve_parameters

convolve_out_2 = DENSITY CONVOLVE 3 >= 0.3 < 0.4 TRUNCATE WINDOW 25 FILE
convolve_parameters

convolve_out_3 = DENSITY CONVOLVE 3 >= 0.4 TRUNCATE WINDOW 25 FILE
convolve_parameters

orig3 {COPY orig3 } DRC CHECK MAP orig3 3
convolve_out_1 {COPY convolve_out_1} DRC CHECK MAP convolve_out_1 301
convolve_out_2 {COPY convolve_out_2} DRC CHECK MAP convolve_out_2 302
convolve_out_3 {COPY convolve_out_3} DRC CHECK MAP convolve_out_3 303
```

### Example 2 (Gaussian Kernel Definitions)

```
litho file dx [
    scale 8.2 gauss 100.000000
    scale -6.50 gauss 1000.000000
    vboasis_path input.oas
]
m = density convolve prime0 < 0.200 >= 0.100000 wrap window 100 inside of \
34695.968000 27518.852 36687.824 29565.584 file dx
```

**Example 3 (Fractal Kernel Definitions)**

```

litho file convolve_param [
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 160 250
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 90 160
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 40 90
    scale 1 fractal 2 rmin 1 rmax 250 convolution_grid 10 partial 1 40
    upsample_interpolation linear
]
d10 = density convolve poly [1-area(poly)/area()] <0.1 window 10 file \
convolve_param

```

**Example 4 (Report)**

```

TIME FOR CONVOLUTION PHASE: CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3
TIME FOR OUTPUT PHASE: CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3
TIME FOR EXPRESSION EVALUATION, SUBSAMPLING, CONVOLUTION, OUTPUT PHASE:
CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3
convolve_out_1 (HIER TYP=1 CFG=1 HGC=1 FGC=1 VHC=F VPC=F)
convolve_out_2 (HIER TYP=1 CFG=1 HGC=3 FGC=3 VHC=F VPC=F)
convolve_out_3 (HIER TYP=1 CFG=1 HGC=5 FGC=5 VHC=F VPC=F)
CPU TIME = 0 REAL TIME = 0 LVHEAP = 1/3/3 OPS COMPLETE = 5 OF 8

```

The HGC value for each output layer shows you the number of geometries created. These geometries are merged representations of all the density windows that met the constraint for the given Density Convolve calculation.

## Device

Device recognition

```
DEvice {element_name [‘(‘model_name‘)’]} device_layer {pin_layer [‘(‘pin_name‘)’] ...}
[‘<‘auxiliary_layer‘>’ ...]
[‘(‘swap_list‘)’ ...]
[BY NET | BY SHAPE]
[NETLIST MODEL netlist_model_name]
[NETLIST ELEMENT netlist_element_name]
[TEXT MODEL LAYER text_layer]
[TEXT PROPERTY LAYER text_layer [text_layer ...]]
[SIGNATURE { “string” | ? } reference_layer { context_layer ... }]
[‘[‘property_specification‘]’]
```

### Summary

The DEvice statement defines how instances of layout devices are to be recognized. For netlisting purposes, this statement names device instances and their pins. It also calculates device properties, as required. For additional information about the handling of layout devices, see the “[Device Recognition](#),” “[LVS Circuit Comparison](#),” and “[SPICE Format](#)” chapters in the *Calibre Verification User’s Manual*.

### Parameters

- *element\_name*

A required schematic component type of the device, which may be built-in or user-defined. Predefined element names specify built-in (or reserved) devices (see Tables 4-5 and 4-6). All other devices are user-defined (or generic). See the “Built-in Device Types” section of the *Calibre Verification User’s Manual* for a complete discussion of this subject. The first parameter within the statement is always taken to be the element name.

Table 4-5 shows built-in devices that have default property computations, hard-coded pin names, and are intrinsic to the device recognition module of a netlist extraction run.

**Table 4-5. Built-in Devices with Default Properties**

Reserved Element Name	Definition	Hard-coded Pin Names <sup>1</sup>	Default Calculated Properties	Default <sup>2</sup> Property Parameters
MN, MP, MD, ME	MOS transistor	G - 1st pin layer S - 2nd pin layer D - 3rd pin layer [B] - optional 4th pin layer	W, L (width & length)	effective_width_factor (default is zero if not supplied)
Example: <b>DEV MP(PMOS) gate gate(G) diff(S) diff(D) [0.5]</b> Here the component type is MP, the model name is PMOS, the device layer is gate, three pin layers are shown, and the effective width factor (for bend compensation) is 0.5.				

**Table 4-5. Built-in Devices with Default Properties (cont.)**

Reserved Element Name	Definition	Hard-coded Pin Names <sup>1</sup>	Default Calculated Properties	Default <sup>2</sup> Property Parameters
D	diode	POS - 1st pin layer NEG - 2nd pin layer [SUB] <sup>3</sup> - optional 3rd pin layer	A, P (area & perimeter)	none
<b>Example: DEV D diode pin_in pin_out</b> Here the component type is D, the device layer is diode, and the POS and NEG pin layers are pin_in and pin_out, respectively.				
C	capacitor	POS - 1st pin layer NEG - 2nd pin layer [SUB] <sup>3</sup> - optional 3rd pin layer	C (capacitance)	capacitance per unit area, capacitance per unit length; defaults are 0 if not supplied
<b>Example: DEV C(POLY) poly_cap anode cathode [1.6 0.07]</b> Here the component type is C, the model name is POLY, the device layer is poly_cap, anode is the POS pin layer, cathode is the NEG pin layer, 1.6 is the capacitance per unit area (unit length squared), 0.07 is the capacitance per unit length. See the <a href="#">Unit Capacitance</a> and <a href="#">Unit Length</a> specification statements.				
R	resistor	POS - 1st pin layer NEG - 2nd pin layer [SUB] <sup>3</sup> - optional 3rd pin layer	R (resistance)	resistivity (default is zero if not supplied)
<b>Example: DEV R int_poly met1(POS) met2(NEG) (POS NEG) [1.1]</b> Here the component type is R, the device layer is int_poly, the POS and NEG pins are met1 and met2, respectively, the POS and NEG pins are swappable, and the resistivity is 1.1 resistance units per square. See the <a href="#">Unit Resistance</a> specification statement.				
Q	bipolar transistor	C - 1st pin layer B - 2nd pin layer E - 3rd pin layer [S] - optional 4th pin layer	none	none
<b>Example: DEV Q(BJT) base coll base emit</b> Here the component type is Q, the model name is BJT, base is the device layer, and there are three pins.				

1 Additional pins may be specified and you must supply the pin names. Specifying more pins than what are shown in the table results in a user-defined device for both device recognition and LVS comparison.

2 Parameters are user-supplied values that determine how default calculated properties are computed.

3 SUB is the default pin name and can be overridden.

You can use the same **element\_name** in more than one Device statement, subject to the following restriction: all statements that have the same **element\_name** and the same number

of pin layers must also have the same set of pin names, and the same swap conditions on the pins. Multiple specifications of an *element\_name* allow alternate methods of laying out the same functional circuit component.

The following Device statements show two examples of defining a capacitor:

```
DEVICE C (CP) diff_and_poly poly diff_not_poly (POS NEG) [1.6 0.07]
DEVICE C (CM) m_cap metall1 metall2 (POS NEG) [1.4 0.095]
```

The first statement defines how poly-diffusion capacitors are recognized. The second defines how metal capacitors are recognized. Previous statements in the rule file (not shown) would have derived the two device seed layers diff\_and\_poly and m\_cap.

The *element\_name* is what LVS uses to match device instances to schematic components. In ICtrace, this corresponds to values of the schematic component property specified with the \$set\_lvs\_component\_type\_properties() function.

Built-in devices for recognition (the ones discussed previously) are also built-in for LVS comparison. This means they participate in device reduction, logic gate recognition, filtering of unused devices, processing of soft substrate pins, and other LVS comparison processes appropriate for the given device. See [LVS Reduce](#), [LVS Soft Substrate Pins](#), [LVS Compare Case](#), [LVS Filter](#), [LVS Recognize Gates](#), and [LVS Split Gate Ratio](#) for more information on related LVS comparison specification statements.

**Built-in devices without default properties** — Device types J (JFET), L (inductor), LDD (generic lightly doped drain MOS), LDDD, LDDE, LDDN, LDDP, M (generic MOS), and V (voltage source) also behave like built-in element types in LVS comparison if they conform to the conventions of Table 4-6. This means they participate in the special LVS processing mentioned in the preceding paragraph. Devices J, L, LDD\*, and V are netlisted as primitive subcircuits even if they conform to the pin naming conventions. There are no associated default calculated properties for these devices.

**Table 4-6. Built-in Devices Without Default Properties**

Reserved Element Name	Definition	Pin Names <sup>1, 2</sup>
J	JFET transistor	G - gate pin layer S - source pin layer D - drain pin layer
L	inductor	POS - 1st pin layer NEG - 2nd pin layer
LDD, LDDD, LDDE, LDDN, LDDP	lightly doped drain MOS	G - gate pin layer S - source pin layer D - drain pin layer [B] - optional bulk pin layer
M	generic MOS transistor	G - gate pin layer S - source pin layer D - drain pin layer [B] - optional bulk pin layer

**Table 4-6. Built-in Devices Without Default Properties**

Reserved Element Name	Definition	Pin Names <sup>1, 2</sup>
V	voltage source	POS - 1st pin layer NEG - 2nd pin layer

1 Additional pins may be specified and you must supply the pin names. Specifying more pins than what are shown in the table results in a user-defined device for LVS comparison.

2 You may choose your own pin names, but this causes the device to be user-defined for comparison.

Note that the devices in Table 4-6 do not have hard-coded pin names. If you choose your own pin names other than the ones shown for these device types, then they are considered user-defined devices for LVS comparison and do not participate in device reduction, device filtering, soft substrate processing, or other special LVS handling that occurs for devices that have built-in comparison algorithms. Device reduction for user-defined devices can be enabled through the generic [LVS Reduce](#) statement. Similarly, device filtering for user-defined devices can be enabled through the generic [LVS Filter](#) statement.

You can specify Device types with arbitrary names as built-in by using the [LVS Device Type](#) statement. Such devices must conform to the pin-naming conventions of [Table 4-5](#) or [Table 4-6](#), depending on the type of device.

- *(model\_name)*

An optional string, which must be enclosed in parentheses, that corresponds to the model property of the schematic. It is also known as the component subtype of the device. If specified, it must immediately follow the *element\_name* parameter. The TEXT MODEL LAYER parameter may override *model\_name*.

It is recommended that you use the following conventions for model names of MOS devices:

- Device MN model names should start with N.
- Device MP model names should start with P.
- Device ME model names should start with E.
- Device MD model names should start with D.
- Device M model names should not start with letters N, P, E, or D.

Devices that do not follow these conventions are represented as subcircuit calls in an extracted SPICE netlist. Primitive subcircuit definitions are generated as needed.

If you do not follow these conventions, device properties may not be traced the way you want. Using [LVS Netlist Allow Inconsistent Model](#) YES can overcome this limitation but is not the recommended practice.

See [LVS Strict Subtypes](#) for information on the matching of devices without subtypes to instances with subtypes.

- ***device\_layer***

A required layer containing the device recognition shapes. It is also called the *seed* layer. The device layer is the first layer you specify within the statement given the recognition precedence of the ***element\_name*** as discussed previously. Device recognition is centered around each device (or seed) shape on this layer. You can specify the ***device\_layer*** as one of the ***pin\_layer*** arguments, but not more than once per statement. Connectivity need not be established on seed layers and it may be undesirable to do so in some cases because of the risk of shorting pin connections through the device body.

- ***pin\_layer***

A required layer on which pin shapes for the device are found. You can specify multiple ***pin\_layer*** names. These layers must have their connectivity established; that is, these layers must have net IDs. A layer carries a net number when one of the following conditions is satisfied:

- A [Connect](#) or [Sconnect](#) operation contains the layer.
- A node-preserving operation derives the layer from a separate layer carrying a net number (see “[Layer Operations](#)” on page 69 for a discussion of node-preserving operations).
- A [Stamp](#) operation transfers net numbers to the layer.

Device recognition logic, not the connectivity extractor, associates pin shapes to device shapes. Therefore, you do not need to connect the pin layers to the device layer by Connect statements. To do so would short the pins together through the device shape.

Pins using the same layer in a Device statement are interchangeable and belong to the same pin-swap group.

Note that pin shapes must touch or overlap device shapes in order for a device to be recognized. Touching at a corner point does not satisfy this condition. Device shapes that do not touch the correct pin layers are considered *bad devices*, and are listed as such in either the circuit extraction report for nmLVS-H or the LVS report in flat nmLVS.

If the ***element\_name*** is built-in for recognition, then you must specify one layer for each pin. The order of the layers must match the pins as listed in Table 4-5. If a device has more than one pin on a given layer, you must repeat that layer as a parameter for each associated pin.

Reserved MOS devices MN, MP, MD, and ME must have at least three pins. The first three pins are hard-coded (G, S, and D; or gate, source, and drain) and default to these values if not supplied. The optional fourth pin layer also has a hard-coded name (B, for bulk), which defaults to this value if you supply a fourth pin layer without a pin name. Note that in the extracted SPICE netlist, pins for these devices are listed in D, G, S, B order.

Reserved capacitors, resistors, and diodes (C, R, and D) must have at least two pins. The first two pin names are hard-coded (POS and NEG, or P and N—for positive and negative), and default to these values if not supplied. The optional third pin does not have a hard-coded name, but defaults to SUB if you supply a third pin layer without specifying a pin name.

Reserved bipolar devices (Q) must have at least three pins. The first three pin names are hard-coded (C, B, and E; for collector, base, and emitter) and default to these values if not supplied. The optional fourth pin has a hard coded name (S, for substrate) and defaults to this value if a fourth pin is specified without a pin name.

In addition, MN, MP, MD, ME, C, R, D, and Q devices can have any number of pins with any user-supplied names in addition to the hard-coded pins; however, such devices are considered user-defined and are netlisted as primitive subcircuits (X calls appear in the netlist). You must specify names for extra pins explicitly. The pin order for the pins that are required for a built-in device must be used. Any additional pins to those that are allowed by the built-in syntax may be specified in any order and pin names must be provided. The pins are then written to the netlist in the order they are specified in the Device statement.

If the *element\_name* is built-in for comparison but not for recognition, then you must match the pins shown in Table 4-6 if you want the device to be treated as built-in for comparison. Otherwise, the device receives no special comparison treatment. If a device has more than one pin on a given layer, you must repeat that layer as a parameter for each associated pin.

If the *element\_name* is user-defined, then the number of *pin\_layer* parameters is unlimited. No *pin\_layer* order is enforced in the Device statement in this case for user-defined *element\_name* parameters. The order of pin layers determines the order in which pins are listed in an extracted netlist. Pins that come from the same layer are interchangeable and belong to the same pin-swap group.

For any device that is treated as user-defined (netlisted as a primitive subcircuit), the order of *pin\_layer* parameters determines the order in which pins are listed in an extracted netlist.

The total number of unique layers appearing in all Device statements sharing the same *device\_layer* is unlimited. However, there may be some performance overhead for using more than 32 unique layers per seed layer in 32-bit Calibre, or more than 64 unique layers per seed layer in 64-bit Calibre. This overhead exists whenever the total number of unique layers appearing in all Device statements sharing any particular seed layer is greater than 32 (64 in 64-bit Calibre). This overhead then exists in *all* Device statements in the rule file, regardless of their seed layers.

The same device, auxiliary, and pin layers can appear in multiple Device statements. However, to prevent ambiguity, the following restriction is enforced: if two statements have the same *device\_layer*, then it must not be possible to reorder the list of auxiliary and pin layers in one statement so that it exactly matches the list of auxiliary and pin layers in the other statement. In making the comparison, the *device\_layer* should be ignored in any list where it appears as a *pin\_layer*. For example:

```
device one A B(p) A(q)  
device two A B(p)
```

causes a compilation error since pin A in one is ignored. This results in a device ambiguity the compiler does not allow.

- *(pin\_name)*

An optional name, enclosed in parentheses, that explicitly names each pin for netlisting.

You can specify the parameter pair ***pin\_layer*** (*pin\_name*) any number of times in one statement.

If the ***element\_name*** in the statement is built-in for recognition, then default *pin\_name* parameters are assigned as shown in Table 4-5. You can specify the ***device\_layer*** as one of the pin layers, but not more than once per statement. You do not need to supply explicit pin names for built-in devices, but if supplied, the names must match the default names that would have been assigned anyway if you want to use the built-in algorithms for property calculation and similar internally-managed computations. Supplying pin names for built-in-for-recognition elements, even though not necessary, makes the rule file more readable.

If the built-in element has other pins in addition to those shown in Table 4-5, then a *pin\_name* parameter must be supplied following each extra *pin\_layer* in the Device statement. The order of the extra pin layers is unimportant in the Device statement. For the extra pins, you can create any pin names you wish; reserved keywords must be enclosed in double quotation marks (""). Providing extra pins to a built-in Device element causes that device to be treated as user-defined. Pin names are then netlisted in primitive subcircuits in the order the pins appear in the Device statement.

If the device element is user-defined for device recognition (devices that are built-in for LVS comparison from Table 4-6 are in this category), then you must supply a *pin\_name* parameter following each *pin\_layer* in the Device statement. You may use any pin names as desired; reserved keywords must be enclosed in double quotation marks ("").

If you have multiple user-defined Device statements corresponding by ***element\_name***, and the pin order differs across these statements (this practice is discouraged), then the first Device statement in the rule file for the element type determines the pin order for the .SUBCKT definition. The pin order for instances of this device element conforms to the .SUBCKT definition, not to the individual Device statements.

You cannot use the same *pin\_name* parameter twice in the same Device statement, but you can reuse it in other statements. If the schematic for the device uses certain pin names, then the same names should be used in the Device statement so that nmLVS applications can match correct pins.

- *<auxiliary\_layer>* [*<auxiliary\_layer>* ...]

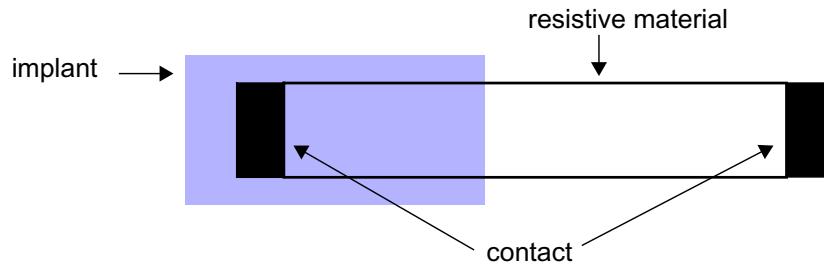
An optional layer name, enclosed in angle brackets (<>), that identifies layers used to classify device instances and participate in property computation. You can specify *<auxiliary\_layer>* any number of times in one statement. They can appear before, after, or intermixed with the pin layers.

An auxiliary layer is a layer containing shapes that are neither seed nor pin shapes. You can use auxiliary layers to interact with other shapes of the device for property alteration or to classify the device. For each auxiliary layer present in a Device statement, at least one shape from that layer must touch or overlap the seed shape before a device instance can be extracted. You cannot limit the number of overlapped shapes, although a property

computation can determine the number of shapes found to touch or overlap by using the COUNT( ) function of the built-in property computation language. See “[Device Property Computation Built-In Language](#)” on page 1776.

Consider an adjustable resistor shown in Figure 4-38. The resistor is formed from a rectangular bar of resistive material with a contact at each end. A variable size implant over one end of the resistor is used to alter the material’s resistivity.

**Figure 4-38. Adjustable Resistor**



The implant layer is an example of an auxiliary layer because it is neither the seed layer nor a pin layer, but it affects the total resistance. The presence or absence of the implant determines whether the resistor is an ordinary resistor or an adjustable resistor. If the implant is present, its geometric relationship to the resistor shape must be available to compute the resistance. If absent, a simpler resistance computation can be performed.

Here are examples:

**DEV R(nonadjust) resist contact(POS) contact(NEG) [resistivity\_value]**

**DEV R(adjust) resist contact(POS) contact(NEG) <implant> [property\_computation]**

You do not need to establish connectivity on the auxiliary layers. Any node information on auxiliary layers is unavailable to device extraction.

An auxiliary layer cannot appear twice in the same Device statement.

You cannot use a layer as both an *<auxiliary\_layer>* and a *pin\_layer* in the same Device statement, nor in any other Device statement using the same *device\_layer*.

- (*swap\_list*)

An optional list of *pin\_name* parameters, in parentheses, that specifies groups of pin names that are interchangeable. You can specify a swap list any number of times in one statement. Each *swap\_list* is of the following form:

**(*pin\_name1* *pin\_name2* ... *pin\_nameN*)**

The pin layers identified by the pin names in a given pin swap group are said to be swappable for the purposes of device recognition. This allows you to place pins from different layers into the same swap group so that their connections become interchangeable during LVS. By default, two pins are in the same swap group only if they appear on the same layer.

You can specify more than one pin swap group in a Device statement, each in its own parenthesized list. Each group must contain two or more pin names. If a pin swap group contains one *pin\_name* from a given *pin\_layer*, then it must contain all names from that layer; the order of pin names within a swap group is unimportant.

In the following examples, the E1 and E2 pins of *element\_name* QE2 are interchangeable because they are on the same layer. An explicit swap list is not required but could have been specified. The two C devices have been specified as having interchangeable pins (POS and NEG). The pin names specified in the swap lists are the default names for the pins of a capacitor. If these swap lists had not been specified, the capacitor pins would not have been interchangeable because they appear on separate layers.

```
DEVICE QE2 (BJT2) base emitt1(E1) emitt1(E2) base(B) coll(C)
DEVICE C (CP) diff_and_poly poly diff_not_poly (POS NEG) [1.6 0.07]
DEVICE C (CM) m_cap metal_1 metal_2 (POS NEG) [1.4 0.095]
```

Calibre allows Device statements with the same element name to have different pin swappability and different pin names, as long as the model names are different. In the two previous examples of C devices, the pin swap conditions could be different since the model names are different.

Pin swappability for built-in devices can be controlled by the [LVS Builtin Device Pin Swap](#) specification statement.

- **BY NET | BY SHAPE**

An optional keyword set that selects the pin recognition method for the device. Two device recognition statements in the rule file with the same *device\_layer* must have the same pin recognition method. The possible choices are:

BY NET — Treats pin shapes on the same layer and connected to the same net as a single pin. This is the default behavior if you do not include a keyword from this set. The pin fill-in algorithm is used to supply missing pins, where applicable.

BY SHAPE — Treats pin shapes as separate pins even when they are on the same layer and are connected to the same net. Also, no fill-in is attempted to supply missing pins. Specifying BY SHAPE can increase runtime significantly and its use is discouraged.

- **NETLIST MODEL *netlist\_model\_name***

This applies only to Calibre xRC applications, the ITrace netlister, and the -nl command line option for flat Calibre nmLVS.

This optional keyword set specifies to use *netlist\_model\_name* when netlisting element model names in SPICE or Lsim; otherwise, the (*model\_name*) parameter is used. These applications use this only for netlisting and not as component subtypes during comparison. Example:

```
DEVICE mn ngate poly(g) ndiff(s) ndiff(d) psub(b) NETLIST MODEL nmos
```

In Calibre xRC, this setting causes the device to appear in the netlist as a model named nmos.

NETLIST MODEL is not used by nmLVS for circuit comparison and is not used for extracted netlists, except for the Calibre -nl option. It is not used by the Query Server, YieldServer, or the calibre -spice option. The text case of the *netlist\_model\_name* is preserved, unless [LVS Downcase Device YES](#) is specified.

- NETLIST ELEMENT *netlist\_element\_name*

This applies only to Calibre xRC, the ICtrace netlister, and the -nl command line option for flat Calibre nmLVS. For ICtrace, it affects SPICE netlists only. It does not affect Lsim netlists.

This optional keyword set overrides the default SPICE element name prefixes during netlisting. The default names are: C, D, J, M, Q, R, and X (X is the netlist name for user-defined devices).

NETLIST ELEMENT is not used by LVS for circuit comparison and is not used for extracted netlists, except for the Calibre -nl option. It is not used by the Query Server, YieldServer, or the calibre -spice option. The text case of the *netlist\_element\_name* is preserved, unless [LVS Downcase Device YES](#) is specified.

In the examples below, LDD devices are netlisted as M devices, and L devices are netlisted as y devices:

```
DEVICE LDD gate sd(D) poly(G) sd(S) bulk(B) NETLIST ELEMENT "m"
DEVICE L seed contact(POS) contact(NEG) NETLIST ELEMENT "y"
```

In addition, a character can be specified to separate the instance name from the device ID number. In the example below, X devices are netlisted as x\$n where *n* is the device ID number:

```
DEVICE X seed a(A) b(B) NETLIST ELEMENT "x$"
```

In Calibre xRC, this setting causes the device to appear in the netlist as the type *netlist\_element\_name*.

- TEXT MODEL LAYER *text\_layer*

An optional keyword set that specifies a layout database text layer that determines the model name for device instances recognized by the Device statement. The *text\_layer* must be an original layer or layer set. If a layer set is used, then all of its member layers are used.

For each device instance, the device recognizer searches for the basepoints of text objects on the specified *text\_layer* that intersect the seed polygon of the device. The tool assigns the value of the intersecting text object as the model name of the device instance. When more than one text object from the specified *text\_layer* intersects the seed polygon, then the device is classified as bad. The equivalence of the text objects determined by string comparison is subject to the setting of [Layout Preserve Case](#).

When you specify both TEXT MODEL LAYER and a (*model\_name*) parameter in a Device statement, the (*model\_name*) parameter serves as default; device instances that do not intersect text from *text\_layer* receive the value of (*model\_name*). If you do not specify a (*model\_name*) parameter, device instances that do not intersect text from *text\_layer* have no model name.

The following are requirements for the use of the TEXT MODEL LAYER optional keyword:

- You can specify the TEXT MODEL LAYER parameter only once per Device statement.
- For Device statements that have the same seed layer and use the TEXT MODEL LAYER optional keyword, you cannot specify a different *text\_layer*. You may have one Device statement with a TEXT MODEL LAYER and another Device statement with the same seed layer but no TEXT MODEL LAYER.

During the reading of layout database text objects for TEXT MODEL LAYER parameters, the tool observes layout area filters and any specified cell exclusion. In Calibre applications, *text\_layer* is subject to editing with the [Layout Rename Text](#) specification statement. The tool reads text objects from TEXT MODEL LAYER parameters throughout the hierarchy in the same manner as shapes. Text objects are read from all layers specified in TEXT MODEL LAYER parameters. [Text Layer](#) and [Text Depth](#) specification statements do not control the *text\_layer* objects read by the tool.

Text objects defined in [Layout Text](#) specification statements behave like database text and participate in the processing of TEXT MODEL LAYER parameters. Text objects defined in [Text](#) specification statements are not used in the processing of TEXT MODEL LAYER.

In hierarchical operation, text objects can be placed in the layout hierarchy at a level above, below, or on the same level as the respective device seed polygons. Text objects interact with seed polygons regardless of their respective hierarchical levels in a manner similar to the interaction between seed polygons and pin polygons. When a device seed polygon and a corresponding text object are at different levels of hierarchy, the seed polygon or text object are promoted for processing up the hierarchy, as necessary, to the level of their lowest common parent cell. Promotion of text objects is an internal event with no side effects; the text object is not removed from its original cell. Seed promotion, on the other hand, forces the tool to form the respective device in the target parent cell rather than in the original cell (subject to device pushdown as described in [LVS Push Devices](#)). See [LVS Show Seed Promotions](#) for related information.

In hierarchical operation, the tool preserves TEXT MODEL LAYER text objects and brings them up the hierarchy as appropriate during cell expansion, such as for [Expand Cell](#) and [Flatten Cell](#) operations, and during automatic expansion of cell placements.

- TEXT PROPERTY LAYER *text\_layer*

An optional keyword set that specifies layout database text layers that may be referenced by [text\\_string\(\)](#) or [text\\_numeric\(\)](#) functions in the Device property computation language. Text values retrieved from those layers by [text\\_string\(\)](#) or [text\\_numeric\(\)](#) may, in turn, be assigned to device instance properties.

Each *text\_layer* argument must be the name or number of an original layer or layer set. If a layer set is specified, then all its member layers are used. At most, one TEXT PROPERTY LAYER specification may appear in a Device statement, although multiple *text\_layer*

parameters may be specified. Multiple text objects on the same layer are allowed as long as they have the same string value.

For each device instance, the device recognizer looks for text objects on the specified *text\_layer* parameters, such that the text location intersects the seed polygon of the device. If a text object is found on a given layer, then any occurrences of the TEXT\_STRING() or TEXT\_NUMERIC() data retrieval functions referencing that layer return the string value or numeric value of the observed text object. If more than one text object on a given layer intersects the seed polygon, then the device is rejected and classified as a *bad* device. (You may have multiple text objects as long as they are on different layers).

As in other statements, reading of layout database text objects for TEXT PROPERTY LAYER parameters observes area filters, any specified cell exclusion and, in Calibre, is subject to editing using [Layout Rename Text](#) specification statements. Text objects for *text\_layer* parameters are read from throughout the hierarchy in the same manner as shapes. Text objects are read from all layers specified in *text\_layer* parameters and are not controlled by [Text Layer](#) or [Text Depth](#) specification statements.

Text objects defined in [Layout Text](#) specification statements behave like database text, and thus participate in the processing of TEXT PROPERTY LAYER parameters. Text objects defined in [Text](#) specification statements are not used in the processing of TEXT PROPERTY LAYER parameters.

In hierarchical operation, text objects may be placed in the layout hierarchy at a level above, below, or the same as respective device seed polygons. Text objects interact with seed polygons regardless of their respective hierarchical levels in a manner similar to the interaction between seed polygons and pin polygons. When a device seed polygon and a corresponding text object are at different levels of hierarchy, then the seed polygon or text object are promoted for processing up the hierarchy, as necessary, to the level of their lowest common parent cell. Promotion of text objects is an internal event with no side effects; the text object is not removed from its original cell. Seed promotion, on the other hand, causes the respective device to be formed in the target parent cell rather than in the original cell (subject to device pushdown as described in [LVS Push Devices](#)). See [LVS Show Seed Promotions](#) for related information.

TEXT PROPERTY LAYER text objects are also preserved and are brought up the hierarchy, as appropriate, during cell expansion, such as in [Expand Cell](#) and [Flatten Cell](#) operations, and during automatic expansion of cell placements.

- SIGNATURE { “string” | ? } *reference\_layer* { *context\_layer* ... }

Optional keyword and parameter set that process device signatures. Usage details for this keyword set are found in the “[Device Signatures](#)” section of the [Calibre Verification User's Manual](#). Use of this keyword set requires a Calibre Advanced Device Properties (ADP) product license. It can be used in Calibre nmLVS and nmLVS-H.

The SIGNATURE keyword set is used to identify a device based upon a set of polygons in the vicinity of the device. The polygons are used as a geometric identification template, called a signature, which is used to match a layout device. This feature is intended to be

used with a “golden library” of reference devices that you intend to match in a layout based upon a signature.

The *reference\_layer* is a single layer that is used to define a region around a device for the purpose of either generating a signature, or for matching a signature. A *reference\_layer* polygon has a logical intersection (that is, a non-empty result from a Boolean AND operation) with the device seed and *context\_layer* polygons. The *context\_layer* polygons that intersect the *reference\_layer* polygon, together with the *reference\_layer* polygon itself, form a geometric pattern that is stored in a signature. There can be more than one *context\_layer* specified with each SIGNATURE keyword set. The *reference\_layer* and *context\_layer* can be any layers of your choosing; however, once a *reference\_layer* is used for signature generation, it may only be used with the same list of *context\_layer* arguments in other Device statements in the rule file. Text on any of these signature layers is ignored.

The “?” character is used in a rule file when generating a set of signatures from a golden library of devices. It is used in conjunction with the -siggen command line option. See the “[Calibre nmLVS/nmLVS-H](#)” section of the *Calibre Verification User's Manual* for details about this option.

A signature is generated in the log file of an LVS or connectivity extraction (calibre -spice) run, and this signature can be saved for later use as a *string* parameter. The section in the log file that contains the generated signature appears similar to this example:

```
DEViCE C (mimc) cap met(POS) met(neg) SIGNATURE
AcZSuedavurxCm8B.C501gI8IJZ:x5gUdvBv0kjEFns2ggb0PecBn:C5ro.FZ1HVevEiqNlN
:BNDmGj4kZoo3Ev:7DJ4g4JEpHi2BnQ632KoAcDeltKFsnbVfJy.LeJUNzrsYv82.2Kwmv2QJ
T1GfdfHM2P32a06F5:tMAVL8do! ref met
HALO SIGNATURE: cell TOPCELL has halo signature of
AcZSuedavurxCm8B.C501gI8IJZ:x5gUdvBv0kjEFns2ggb0PecBn:C5ro.FZ1HVevEiqNlN
:BNDmGj4kZoo3Ev:7DJ4g4JEpHi2BnQ632KoAcDeltKFsnbVfJy.LeJUNzrsYv82.2Kwmv2QJ
T1GfdfHM2P32a06F5:tMAVL8do!
```

The signature string is lengthy and must be copied exactly as it appears in the log file, with no line breaks.

The *string* parameter is an alphanumeric string enclosed in quotes. This string is generated as described in the previous paragraphs. It is used in place of the “?” character when you want to extract devices by their signatures. This string is decoded at runtime and is used to check that a device has the proper orientation of polygons surrounding it in order to determine if the device has the proper signature (the -siggen command line option is not needed for this scenario). Note that the order of *context\_layer* arguments used in a signature matching run must be identical to what was used in the signature generation run, or the signature match will fail.

Signatures are portable between 32- and 64-bit machines, but the *reference\_layer* geometry must have dimensions that are representable in 32-bits of coordinate space, even on a 64-bit machine.

- ‘[*property\_specification*]’

An optional parameter, enclosed in square brackets ([ ]), that specifies computation of properties for device instances.

A property specification can take one of two forms: either a list of floating-point numbers that are used in default property computations performed internally by the tool, or a short program written in a property computation language. The list of numbers form is only applicable to reserved element names as shown in [Table 4-5](#).

See the “Default Property Computations” section in the [Calibre Verification User’s Manual](#) for a complete discussion of default property computations. The default property computations for built-in devices are shown later in this section under [Examples](#).

A user-specified program overrides default property computations for built-in devices (and makes them user-defined), or provides property computations for custom properties for user-defined devices.

The following is an example of each form of property specification:

```
DEVICE R resistor_layer metal_1 metal_2 [1.1]
// resistivity is 1.1 resistance units per square
// internal default calculation is used to find resistance

DEVICE R layer metal_1 metal_2
[
property R
    R = .5*(AREA(layer)-AREA_COMMON(layer, metal_1)-
    AREA_COMMON(layer, metal_2))
]
/* R is defined as the resistance of the device and
calculated as shown */
```

If you provide a property specification program, you take on responsibility for calculating *all properties* of the device, even if it is a built-in type. Failure to do so properly can cause unexpected results in device reduction, circuit comparison, property tracing, and so forth. The built-in language for device property computation is discussed under “[Device Property Computation Built-In Language](#)” on page 1776.

Property names must begin with a letter character followed any legal SPICE characters. See “[General SPICE Syntax Summary](#)” in the [Calibre Verification User’s Manual](#) for more details.

Certain user-defined properties for C and Q devices are coded as comments by default. See [LVS Netlist Comment Coded Properties](#) for details.

Properties for devices can be computed using [DFM Property](#) and attached using the [Device Layer ANNOTATE](#) option. The [LVS Annotate Devices](#) specification statement can be used for netlisting of such properties.

See the “[Macros](#)” section in Chapter 2 for information on defining property computations in a macro.

See [Trace Property](#) for how to trace device property values during LVS. Note that the layout property names you specify in a Trace Property statement should come from the (default or user-defined) property computations in a Device statement when Calibre is used for extracting the layout netlist, not the property names that appear in your SPICE netlist. See [Table 4-5](#) on page 196 for a listing of the default property names available for tracing.

## Description

This statement names and classifies a device; tells the tool how to recognize instances of the device; names the pins and their swap groups explicitly or by default; and in certain cases provides the constants used in computing property values.

Devices that are classified in the layout are written to an extracted layout netlist.

The basic process of device recognition is this:

1. Each layer that appears in one or more DEvice statements is scanned together with all relevant pin and auxiliary layers.
2. For each device shape, an initial set of connected pins and auxiliary shapes is identified.
3. If the resulting pattern of shapes matches one of the Device statements, the device is classified.
4. For device shapes that are not classified by step 3, the pin-fill algorithm attempts to find missing pins to obtain a unique match.
5. If step 4 fails, the device shape is classified as ill-formed, or bad.
6. Properly recognized devices have their properties computed and assigned, as necessary.

Consider the following Device statements:

```
Dev D1 device_layer pin_layer1(A) pin_layer2(B)
Dev D2 device_layer pin_layer1(A) pin_layer2(B) pin_layer2(C)
Dev D3 device_layer pin_layer3(D) pin_layer4(E)
```

For the given set of Device statements, a device shape on device\_layer that touches pin\_layer1, pin\_layer2, and pin\_layer3 is classified as bad. However, if the third statement were not present, the device shape would be classified as D1. The reason is only pin\_layer1 and pin\_layer2 would be read in for device classification; pin\_layer3 would be ignored.

For a complete description of the device recognition process, see “Recognition Logic” in the *Calibre Verification User’s Manual*.

For a description of LVS connectivity extraction, see “[LVS Connectivity Extraction](#)” in the *Calibre Verification User’s Manual*.

## Name Selection

You should choose user-defined device element, subtype, and property names that are SPICE compliant. See “[General SPICE Syntax](#)” in the *Calibre Verification User’s Manual* for additional information. Failure to choose compliant names can result in errors reported by the SPICE parser.

## Restrictions

1. Two Device statements in the same (or Included) rule file with the same *element\_name*, same optional (*model\_name*), and the same number of *pin\_layer* parameters must have the same set of *pin\_name* parameters and pin-swap conditions.

2. A *device\_layer* may appear as only one *pin\_layer* in a Device statement.
3. Two Device statements in the same (or Included) rule file with the same *device\_layer* may not have the same set of *pin\_layer* and *auxiliary\_layer* parameters. The *pin\_layer* order is ignored, but the number of occurrences is not.
4. Two Device statements in the same (or Included) rule file with the same *device\_layer* must both have either BY NET or BY SHAPE specified.
5. A *pin\_name* may appear only once in a Device statement.
6. An *auxiliary\_layer* may appear only once in a Device statement and may not be identical to the *device\_layer*.
7. An *auxiliary\_layer* may not coincide with a *pin\_layer* in any Device statement that shares the same *device\_layer*.
8. A *pin\_name* may appear only once in a *swap\_list*.
9. A *swap\_list* containing the *pin\_name* of a given *pin\_layer* must contain all *pin\_name* parameters for that *pin\_layer* within a Device statement.
10. Two Device statements in the same (or Included) rule file with the same *device\_layer* must have the same *text\_layer* parameters specified, if any.

If a *device\_layer*, *pin\_layer*, or *auxiliary\_layer* is derived from a DFM Property operation, this requires a *Calibre Advanced Device Properties (ADP) product* license during connectivity extraction.

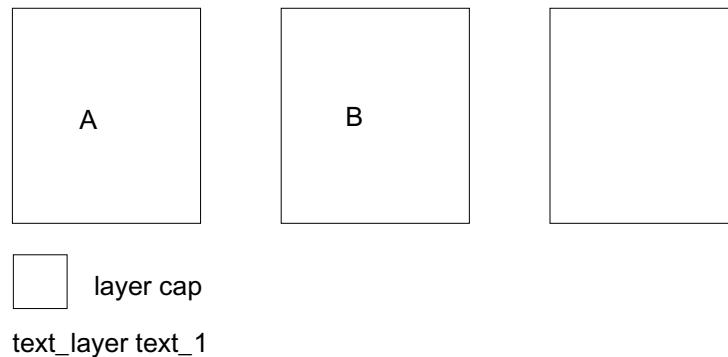
**ICverify considerations** — ICtrace matches the element name to the value of the Design Architect schematic component property specified in the [LVS Component Type Property](#) specification statement (\$set\_lvs\_component\_type\_properties() function in ICtrace). They also match the model name to the value of the Design Architect schematic component property specified in the [LVS Component Subtype Property](#) specification statement (\$set\_lvs\_component\_subtype\_property() function).

## Examples

### Example 1

Figure 4-39 shows three polygons on layer cap, which serves as seed layer for capacitor devices. Two of the polygons have text objects (A and B) on layer text\_1, the TEXT MODEL LAYER. For clarity, pin polygons are not shown.

**Figure 4-39. TEXT MODEL LAYER Example**



Given the statement:

```
DEVICE C cap pos1 neg1 TEXT MODEL LAYER text_1
```

the first capacitor has model name A, the second capacitor has model name B, and the third capacitor has no model name.

Given the statement:

```
DEVICE C(X) cap pos1 neg1 TEXT MODEL LAYER text_1
```

the first capacitor has model name A, the second capacitor has model name B, and the third capacitor has model name X.

### Example 2

These are the default property calculations for the built-in devices. Using [Macros](#) often makes coding the rule file more efficient.

#### Diodes (D) —

```
DEV D diode_layer anode(POS) cathode(NEG)
[
    property A,P
    A = area(diode_layer)
    P = perim(diode_layer)
]
```

**Capacitors (C) —**

```
DEV C cap_layer anode(POS) cathode(NEG)
[
    property C
    CA = area_capacitance      //user supplied
    CP = perimeter_capacitance //user supplied
    CCA = CA * (unit_cap() / (unit_len() * unit_len()))
    CCP = CP * (unit_cap() / unit_len())
    C = CCA * area(cap_layer) + CCP * perim(cap_layer)
]
```

**Resistors (R) —**

```
DEV R res_layer pos_pin(POS) neg_pin(NEG)
[
    PROPERTY R
    RSQ = resistance_per_unit_square //user supplied
    RR = RSQ * unit_res()
    W = .5*(perim_co(POS,res_layer)
            + perim_in(POS,res_layer)
            + perim_co(NEG,res_layer)
            + perim_in(NEG,res_layer))
    R = RR * area(res_layer) / (W * W)
]
```

**MOS transistors (MD, ME, MN, MP) —**

```
DEV MP seed gate(G) diff(S) diff(D)
[
    property W,L
    WEFFECT = effective_width_constant //user-supplied
    W = .5*(perim_co(S,seed) + perim_in(S,seed)
            + perim_co(D,seed) + perim_in(D,seed))
    L = AREA(seed) / W
    if ((BENDS(seed) != 0) && (WEFFECT != 0)) {
        if (W > L) W = W - WEFFECT * bends(seed) * L
        else         L = L - WEFFECT * bends(seed) * W
    }
]
```

See “[Device Property Computation Built-In Language](#)” on page 1776 for details about the device property computation language.

**Example 3**

This code shows the calculation of MOS properties using a macro.

```
//derived pin layers

sd_rsl = sd NOT contact
sd_rs2 = sd_rsl COIN EDGE gate
sd_rs3 = sd_rsl COIN EDGE contact
// the following is an auxiliary layer for device recognition
sd_rs = INT sd_rs2 sd_rs3 < sd_to_gate PARALLEL OPPOSITE REGION

// the sd_rs layer created is necessary for the NRD and NRS
// calculations in M devices.
// sd_to_gate is a variable that stores the largest distance a
```

```

// source/drain contact would ever be from the edge of a gate in your
// process.
// Following is the DMACRO that does the actual calculation.

DEVICE MP pgate pgate(G) psd(S) psd(D) bulk(B) <sd_rs>
  CMACRO m_props sd_rs
DEVICE MN ngate ngate(G) nsd(S) nsd(D) bulk(B) <sd_rs>
  CMACRO m_props sd_rs

DMACRO m_props sd_rs
{
  [ property W, L, AS, AD, PS, PD, NRD, NRS
    weffect = 0

  // CALCULATE W & L
  W = (perim_co(S, G)+ perim_co(D, G)) / 2
  L = area(G) / W
  if ( ( bnds(G) != 0 ) && ( weffect != 0 ) ) {
    if ( W > L ) {
      W = W - weffect * bnds(G) * L
    }
    else
      L = L - weffect * bnds(G) * W
  }
  //CALCULATE AS, AD, PS, PD
  AS = area(S) * (W / perim_in(S, diff))
  AD = area(D) * (W / perim_in(D, diff))
  PS = perimeter(S) * W / perimeter_inside(S, diff)
  PD = perimeter(D) * W / perimeter_inside(D, diff)

  // This section of the property computations measures Number of
  // Resistance Squares in Drain and Number of Resistance Squares in Source,
  // as a first order approximation.

  // The following calculations use edges of contacts instead of centers
  // of contacts. That is,
  // NRD = average distance from gate to contacts' nearest edges / W
  // NRS is similar.

  // The following calculations account for relative placement of contacts
  // to gate and to each other, with the single exception that contacts that
  // have no edges facing the gate edge are not involved in the calculation.

  // Calculations assume all contacts are equally sized.

  SUM_S_LENGTH = perim_in(sd_rs, S) - perim_co(sd_rs, S)
  COUNT_S = trunc((count(sd_rs) * perim_co(sd_rs, S) /
                  perim_co(sd_rs, G)) + 0.5)
  if (COUNT_S != 0) {
    NRS = SUM_S_LENGTH / COUNT_S / W / 2
  }
  else
    NRS = AS / (W * W)

  SUM_D_LENGTH = perim_in(sd_rs, D) - perim_co(sd_rs, D)
  COUNT_D = count(sd_rs) - COUNT_S
  if (COUNT_D != 0) {
    NRD = SUM_D_LENGTH / COUNT_D / W / 2
  }
}

```

```
    else
        NRD = AD / (W * W)
    }
] // end of property calculation
} // end of DMACRO
```

# Device Layer

Layer operation

**DEVICE LAYER** *element\_name* [‘(‘*model\_name*‘)’]  
[ANNOTATE *layer* [‘[‘*device\_annotation\_program*‘]’] [EXCLUDE UNUSED]  
[PROPERTY]

Used only in Calibre nmLVS/nmLVS-H, Calibre YieldAnalyzer, and ICtrace.

## Parameters

- ***element\_name***  
A required component type of the device that controls pin layer count and names for built-in devices. Certain predefined element names specify built-in (or hard-coded) devices (see Table 4-5). All other devices are user-defined or generic devices.
- **(*model\_name*)**  
An optional string, enclosed in parentheses, corresponding to the MODEL property of the device. It is also known as the subcomponent type of the device.
- **ANNOTATE *layer***  
An optional keyword and layer argument that instruct the tool to output a layer that is annotated with DFM properties present on the input *layer* argument. This option useful for attaching complex properties to devices without affecting LVS comparison. The keyword enables use of the output layer with the [LVS Annotate Devices](#) specification statement.
- **‘[‘*device\_annotation\_program*‘]’**  
An optional user-defined program that specifies how to annotate devices on the ANNOTATE layer. The program appears between literal square brackets. See “[Device Annotation Built-In Language](#)” on page 1842 for complete details.
- **EXCLUDE UNUSED**  
An optional keyword that instructs the tool to exclude unused devices from the result layer according to the specified unused device filters. Unused nets are connected only to unused devices. Unused devices are specified with [LVS Filter Unused Option](#) or equivalent LVS Filter Unused statements. All options for LVS Filter Unused Option are supported except the INV, P, and Q options.
- **PROPERTY**  
An optional keyword that specifies to annotate device properties as DFM properties to the result output. For information about DFM properties, see “[DFM Properties](#)” in the *Calibre YieldAnalyzer and YieldEnhancer Reference Manual*.

## Description

This operation creates a layer containing seed shapes from all device instances with the specified *element\_name* and optional *model\_name*. Seed shapes are those from *device\_layer* parameters in [Device](#) operations.

If you do not specify a *model\_name*, then all devices with the specified *element\_name* (with any model name or no model name) are processed. If more than one Device operation matches a Device Layer operation, then the results are merged together by an [OR](#) operation.

The Device Layer operation does not preserve node numbers from input shapes. The result layer of a Device Layer operation cannot be used in [Connect](#), [Sconnect](#), or Device operations, and cannot be used to derive layers for those operations. The Device Layer operation is often used as part of ERC.

In Calibre nmDRC/nmDRC-H and ICrules, the Device Layer operation produces an empty result layer and generates a warning.

## ANNOTATE Keyword

The ANNOTATE option is useful for attaching properties computed using the [DFM Property](#) operation to output layers. The *layer* argument can be any layer generated by DFM Property. This option can be used in conjunction with the LVS Annotate Devices specification statement for device property annotation.

The ANNOTATE keyword supports the optional *device\_annotation\_program*. This is a user-defined program that defines how annotations are made. This capability is useful for adding measurements like stress effects that have the potential to disrupt the hierarchy of more exotic devices. See “[Writing a Device Annotation Program](#)” on page 1842 for a discussion of these programs.

The section “[Using DFM Property with Device Layer ANNOTATE Programs](#)” on page 1848 for a complete example.

## PROPERTY Keyword

Without the PROPERTY keyword, a Device Layer operation can refer to multiple Device statements and the result layer contains seed polygons of all matching devices. When the PROPERTY keyword is used, the Device Layer operation must refer to exactly one [Device](#) statement, otherwise a compiler error is output. Both string and numeric properties are output as DFM properties when this keyword is specified.

An *element\_name* may be specified without a *model\_name* if all models of the device generate identical properties in the same order. When it is used this way, the output layer contains shapes that represent all of the recognized devices regardless of model name. All output shapes are annotated with device properties.

In the following example, the Device Layer operation is used to find all MP devices connected to VCC through the PSD pin layer. The seed polygons of these devices are written out into an RDB together with device properties.

```
DEVICE MP PGATE PGATE PSD PSD PWELL
  [ PROPERTY L, W, AS, AD
```

## Device Layer

---

```
W = PERIMETER_COINCIDE(PGATE, PSD)/2
L = AREA(PGATE)/W
AS = AREA(S)
AD = AREA(D)
]
MP_VCC = ( DEVICE LAYER MP ) INTERACT ( NET PSD "VCC" )
ERC SELECT CHECK MP_CHECK
MP_CHECK {
    DFM RDB MP_VCC "mp_vcc.rdb" ALL CELLS CHECKNAME "MP - Properties"
}
```

Here is a sample RDB file mp\_vcc.rdb generated by these statements. Notice the device properties l, w, as, and ad.

```
top 1000
MP - Properties
2 2 1 Apr 25 18:53:32 2007
IL: CHECK1::mp_p
p 1 4
CN top 1 0 0 1 0 0 1
l 4e-07
w 5e-07
as 1.5e-13
ad 1.5e-13
500 2300
900 2300
900 2800
500 2800
p 2 4
l 2e-07
w 5e-07
as 2e-13
ad 2e-13
2200 2300
2400 2300
2400 2800
2200 2800
```

## Examples

### Example 1

The following example finds all MP devices connected to VCC through the PSD pin layer. Equivalent to ECONNECT MOS[P] PSD CONN VCC OUTPUT ECONN 50 in Dracula.

```
ECONN50 { ( DEVICE LAYER MP ) INTERACT ( NET PSD "VCC" ) }
ERC SELECT CHECK ECONN50
```

### Example 2

The following example finds all MP devices not connected to VCC through the PSD pin layer. Equivalent to ECONNECT MOS[P] PSD DISC VCC OUTPUT EDISC 50 in Dracula.

```
EDISC50 { ( DEVICE LAYER MP ) NOT INTERACT ( NET PSD "VCC" ) }
ERC SELECT CHECK EDISC50
```

**Example 3**

The following example finds all nets that do not connect to MP devices through the PSD pin layer. Outputs all PSD shapes on those nets. Similar to ELCOUNT MOS[P] PSD EQ 0 OUTPUT ELDISC 50 in Dracula.

```
ELDISC50 {  
    NET AREA RATIO PSD ( PSD INTERACT ( DEVICE LAYER MP ) ) == 0  
}  
ERC SELECT CHECK ELDISC50
```

**Example 4**

The following example finds all nets that connect to MP devices through the PSD pin layer. Outputs all PSD shapes on those nets. Similar to ELCOUNT MOS[P] PSD GT 0 OUTPUT ELCOMP 50 in Dracula.

```
ELCONN50 {  
    NET AREA RATIO PSD ( PSD INTERACT ( DEVICE LAYER MP ) ) > 0  
}  
ERC SELECT CHECK ELCOMP50
```

**Example 5**

See “[Using DFM Property with Device Layer ANNOTATE Programs](#)” on page 1848 for an example showing the ANNOTATE keyword.

# DFM Analyze

Layer operation

```
DFM ANALYZE layer [layer ...] [‘[‘expression‘]’ constraint]
[RDB [ONLY] [DV] [DX] [COORD] filename]
[COMMENT “comment_string”]
[ANNOTATE ‘[‘ “name” = “value” ‘]’ ...]
[INSIDE OF {{x1 y1 x2 y2} | EXTENT | LAYER name}]
{ [WINDOW {w | x y} ] { [STEP {s | x y}] |
[NUMWIN { n | nx ny } [OVERLAP { o | ox oy } ]] }
[CENTERS [mx [my]]] [BACKUP] ]
[BY CELL [NOPSEUDO] [NOHIER] [ALLDEPTH] [RESULT CELLS cell_list]]
```

Used only in Calibre YieldAnalyzer applications.

## Summary

Layer operation that is used to identify areas on a layout where manufacturing yield can potentially be improved. Specifically, it provides output of layout regions that meet specified constraints for various layout properties. These properties include area, perimeter, length, and count.

## Parameters

- *layer* [*layer* ...]

A required original layer, derived polygon layer, derived edge layer, or derived error layer (DFM Analyze accepts derived error layers as input for use in certain [Measurement Functions](#).) More than one input layer can be specified.

- ‘[‘*expression*‘]’

An optional expression that performs a calculation involving layer data. The *expression* may involve numbers (including numeric variables), [Unary Operators](#) (+, -, !, ~), [Binary Operators](#) (^, \*, /, +, -, &&, ||), parentheses ( ), variables, [Algebraic and Transcendental Functions](#), [DFM Functions](#), [DFM Properties](#), [Conditional Expressions](#), and most [Measurement Functions](#).

The usual rules of operator precedence apply, and parentheses may be used to establish precedence in the calculation.

- The *expression* must be enclosed in brackets [ ].
- If an *expression* is not specified, the default is:

$$[f_1(layer_1) + f_2(layer_2) + \dots + f_N(layer_N)]$$

where:

- $f_i$  is  $\text{AREA}(layer_i)$  if  $layer_i$  is an original or derived polygon type
- $f_i$  is  $\text{LENGTH}(layer_i)$  if  $layer_i$  is a derived edge layer

- $f_i$  is  $\text{COUNT}(layer_i)$  if  $layer_i$  is a derived error layer
- If an *expression* is not specified, the *constraint* defaults to “ $\geq 0$ ”.
- Division by 0 is defined not to satisfy any constraint.

For more information on expressions, refer to [DFM Expressions](#).

- *constraint*

A numeric constraint interpreted in user units that must be specified with an *expression*. The value specified in the constraint can be positive or negative. The default is “ $\geq 0$ ”.

- RDB [ONLY] [DV] [DX] [COORD] *filename*

Optional keyword set that instructs the tool to create a [Scores RDB](#) with the given *filename*. Within the *filename*, the [Pattern Substitution](#) facility replaces all occurrences of the pattern “%\_t\_” with the name of the top cell for the design.

When ONLY is specified, results are sent only to the RDB. No geometric data is sent to the usual nmDRC results database.

You control exactly which data is written to the RDB through the optional keywords DV, DX, COORD:

- DV — the value of the DFM Analyze expression per window (DV property).
- DX — the values of sub-expressions, such as the DA properties for AREA measurement.
- COORD — window coordinates for the capture window.

By default, the operation writes all these types of data to the RDB. There is no keyword for instructing the operation to write no data to the RDB, as this would result in a meaningless RDB.

Note that if you create an RDB with partial data, you may not be able to use that RDB as input to some downstream tools. For example, Calibre RVE requires that COORD (window coordinates) be present in the RDB.

- COMMENT “*comment\_string*”

An optional keyword and argument used to add comments to the RDB. When present, the operation writes *comment\_string* as check text in the RDB (see [RDB Formats Overview](#)). May be specified multiple times. Each comment adds one line of check text to the appropriate rule check(s).

The *comment\_string* must be a single string, so comments containing spaces or special characters must be enclosed in double quotes (“ ”). While required only for comments containing spaces, double quotes are recommended for all comments.

- ANNOTATE '[' “*name*” = “*value*” ‘]

An optional keyword and arguments used to associate meta data with the \_detail layer that is written to a DFM database. The \_detail layer is produced in addition to the output layer

normally produced by the operation. Unlike the output layer normally produced, which contains merged geometries, the \_detail layer contains unmerged geometries. This keyword is processed only when running Calibre with the -dfm switch. It is ignored when running Calibre with the -drc switch.

Within the arguments for this keyword, the following constraints apply:

- The square brackets [ ] are required.
- The *name* can be a quoted or unquoted string, and need not be unique.
- The *value* must be a quoted string.

You can specify this keyword as many times as needed. Each occurrence adds another annotation (*name* and *value* pair) to the layer.

- INSIDE OF { {*x1 y1 x2 y2*} | EXTENT | LAYER *name* }

Optional keyword set that defines a rectangular boundary within which a data capture WINDOW moves. If you do not specify this optional keyword set in the statement, the default boundary is the extent of the layout data read in for the run.

*x1 y1 x2 y2* — Specifies the corners of a rectangle in user units. If a value is negative, it must be enclosed in parentheses ( ).

EXTENT — Specifies that the boundary is the extent of the input *layer*.

LAYER *name* — Specifies that the boundary is the area of the polygonal extents of the *name* layer (not the polygons themselves). This layer must be an original or derived polygon layer.

- WINDOW {*w* | *x y*}

Optional keyword and parameter set that specifies a data capture window within which the DFM checking is to occur. The default window is the database extent. The WINDOW keyword may be specified separately from the RDB keyword in this operation (this is different from other DFM operations). The following conditions apply when using the WINDOW keyword:

- If you specify WINDOW without NUMWIN, the STEP keyword may be used.
- If you specify both WINDOW and NUMWIN, the STEP keyword may not be used, and the WINDOW size is the minimum acceptable window size. If the window size determined by the NUMWIN setting alone is less than the specified WINDOW size, the window size used is the smallest window size that is both greater than or equal to the specified WINDOW size and evenly divides the layer extent.
- You cannot specify WINDOW with BY CELL.

For details of how data capture windows work refer to [Partitioning by Window](#).

- **STEP {*s* | *x* *y*}**

Optional keyword and parameter set that specifies the fixed distance a window moves within a given boundary. You may specify STEP only if you specify WINDOW and do not specify NUMWIN.

- **NUMWIN { *n* | *nx* *ny* }**

Optional keyword and parameter set that specifies the number of data-capture windows; *n* (or *nx* and *ny*) must be an integer  $\geq 1$ . NUMWIN cannot be specified with BY CELL.

- **OVERLAP { *o* | *ox* *oy* }**

Optional keyword and parameter set that specifies the fractional amount by which a data capture window overlaps the previous data-capture window; *o* (or *ox* and *oy*) must be  $\geq 0$  and  $< 1$ . OVERLAP can only be used if NUMWIN is also specified. If NUMWIN is specified but OVERLAP is not, the default value for OVERLAP is 0.

- **CENTERS [*mx* [*my*]]**

Optional keyword that generates an output layer containing window marker geometries annotated with DFM properties containing the values of the DFM Analyze expression, in addition to coordinates of the data capture window represented by each output polygon. This layer is generated instead of the regular output layer of the DFM Analyze operation.

When this keyword is specified, the output layer of the DFM Analyze operation contains one rectangle for each window that satisfies the DFM Analyze constraint, or for every window if the constraint is not specified. The size of the rectangle can be specified by the optional parameters *mx* and *my*, which determine the size of the output rectangle in horizontal and vertical directions, respectively. If only *mx* is specified, then *my* is assumed to be equal to *mx* and squares are generated. If marker dimensions are not specified, small marker squares of 4\*4 database units are generated.

The rectangles on the output layer generated by a DFM Analyze operation with CENTERS specified cannot abut or overlap. The specified size of the marker rectangle is reduced if necessary to prevent adjacent rectangles from overlapping. Thus, the size of the rectangles cannot exceed the size of the window or the step. Since the windows that touch the top and right boundaries of the chip may be reduced if they do not completely fit inside the boundary, the corresponding marker rectangles may be smaller as well. For example, if the WINDOW keyword is specified, the default STEP is used, and the CENTERS value is larger than the WINDOW size, the marker size is reduced from the specified value and the output layer will contain separate non-overlapping rectangles with narrow gaps between them.

All geometries generated by the DFM Analyze operation with the CENTERS keyword are in the top cell.

The output layer is annotated with DFM properties for all values computed by the DFM Analyze operation. The property named “DV” is always present on the output layer. In addition, other properties normally reported in the RDB created by the DFM Analyze operation are present, for example, the property “DA” is present if the AREA(layer) measurement function is computed.

The output layer is also annotated with DFM properties containing original coordinates of the data capture window represented by each output polygon. These properties have integer values and cannot be referenced with the PROPERTY() function in DFM expressions. They can, however, be accessed using YieldServer.

The CENTERS keyword may be used only with WINDOW, not with BY CELL.

- **BACKUP**

Optional keyword that specifies that if a window overlaps the right or top edges of the design extent, the window is shifted left or down until it no longer overlaps the design extent. This keyword is similar to the BACKUP keyword for the [Density](#) operation.

In WINDOW mode, by default, the data capture windows that extend outside of the design extent are truncated so that their sizes are less than the value specified by the WINDOW option. When BACKUP is specified, the data capture windows that cross the right edge of the design extent are instead shifted to the left until their right edges line up with the right edge of the design extent. Similarly, the windows that cross the top edge of the design extent are shifted down until their top edges line up with the top edge of the design extent. The window in the right top corner can be shifted in two directions at once. As a result, all windows have the same size; however, the shift of the backed-up windows is smaller than the specified STEP value.

- **BY CELL**

Optional keyword that specifies that data is to be partitioned on a per-cell basis. The BY CELL keyword may be specified separately from the RDB keyword in this operation (this is different from other DFM operations). BY CELL cannot be specified when using WINDOW or NUMWIN. If neither this keyword, nor the WINDOW keyword is explicitly specified, then the default WINDOW behavior is used.

In flat Calibre applications, BY CELL causes the DFM operation to evaluate a single capture defined by the total rectangular extent of the input layers for the operation. No further adjustments are made for flat applications.

- **NOPSEUDO**

Optional keyword that specifies that pseudocells created by Calibre during the run are not part of the output. This can only be specified with the BY CELL keyword. Calibre creates pseudocells on an as-needed basis as part of its internal optimization of hierarchy management. These pseudocells appear in the output by default. This keyword causes any results in a pseudocell to be expanded up to the next level of hierarchy.

- **NOHIER**

Optional keyword that affects the processing of geometries that have hierarchical overlap. When NOHIER is specified, these geometries are processed in each cell independently, ignoring cells above and below. When NOHIER is not specified, geometries in a cell that is part of a larger flat geometry contribute to measurements only partially. For COUNT(), the geometries are counted only in the cell in which the entire geometry is assembled (and not in any lower cells). For other measurement functions, only the part of the geometry that is in

the cell, and not overlapped by above geometries, is counted. If there is no hierarchical overlap between geometries in different cells, NOHIER has no effect.

This keyword can only be specified with the BY CELL keyword.

- **ALLDEPTH**

Optional keyword that affects calculation of measurement functions. When ALLDEPTH is specified, the measurement functions for all placements in a given cell are added to the measurement functions for that cell. This keyword can result in the same geometries being counted numerous times. However, the results for the top cell (and similarly, any lower cell) match the results of the design when flattened. When ALLDEPTH is not specified, measurement functions in a cell do not include results from lower cells.

This keyword can only be specified with the BY CELL keyword.

ALLDEPTH and NOHIER can be specified together. In this case, each geometry is measured in the cell in which it resides, then the results are summed up through the hierarchy. If there is overlap of geometries across the hierarchy, geometries are counted multiple times to a greater extent compared to cases where only ALLDEPTH is specified.

- **RESULT CELLS *cell\_list***

Optional keyword that controls expansion of results into a set of result cells specified by *cell\_list* (where the *cell\_list* is defined by a [Layout Cell List](#) statement). When this keyword is used, results are partially flattened in such a way that only the specified result cells contain results. Results from any cell not listed as a result cell are expanded into the nearest containing result cell. The top cell is always considered to be a result cell even if it is not explicitly specified as such, and so all results that are not contained within one of the specified result cells are reported in the top cell. This keyword implicitly activates the NOPSEUDO option, since pseudo cells cannot be specified as result cells.

Example:

```
LAYOUT CELL LIST rcells "A*" "B*"
DFM ANALYZE metall1 BY CELL RESULT CELLS rcells [ AREA(metall1) ] > 0
```

Note that Calibre performs extensive hierarchy transformations, and there is no guarantee that any specified result cell is still present after these transformations. The [Layout Preserve Cell List](#) specification statement, which also uses the Layout Cell List facility, can be used to suppress expansion of the result cells. However, enforcing preserved cells can have a significant impact on Calibre performance.

## Description

Constructs a derived polygon layer consisting of regions such that the *expression* meets the given *constraint*. The output layer may be empty.

DFM Analyze is one of the few SVRF operations that accepts [derived error layers](#) as input. DFM Analyze accepts this type of data as input for use in [measurement functions](#) such as COUNT, EC, and EW.

For a complete list of measurement functions that can be used with DFM Analyze, refer to the [Calibre YieldAnalyzer and YieldEnhancer Reference Manual](#).

### Concurrency

DFM Analyze operations are performed concurrently if they share the following:

- The same input layers (in the same order).
- The same WINDOW options:
  - Same window coordinates and step or same number of windows and overlap.
  - Same INSIDE OF settings.

### RDB Output

RDB output is optional, but helpful in most cases where a detailed analysis of the results is desirable. RDB specification is explained in detail in the [Results Databases](#) section.

### Limitations

For input layers containing skew edges, AREA values can differ between flat and hierarchical runs. When using derived error layers as input, the COUNT values can differ between flat and hierarchical runs.

### Examples

#### Example 1

Suppose you are concerned with the average area of metal polygons in your layout database, and you want to output 100 x 100 um regions where the average area per polygon is below a certain value “z.”

```
avg_polygon_area {
    DFM ANALYZE metal WINDOW 100
    [AREA(metal) / COUNT(metal)] < z
}
```

How this rule check works:

- The operation uses a 100 x 100 um data collection window (triggered by WINDOW 100) and scans the metal layer. The exact details of how this scan occurs are covered under [Partitioning by Window](#).
- The AREA(metal) function calculates the area of the metal layer in the capture window.
- The COUNT(metal) function counts the number of metal polygons enclosed by the window.
- The entire expression in the brackets [ ] gives an idea of the average area of polygons in the window.
- Windows with an average metal polygon area less than “z” are output as check results.

If an expression denominator is 0, the entire expression is undefined because division by 0 is undefined. In this case, the expression does not satisfy the constraint. For example:

```
[AREA(metal) / (COUNT(metal)) < z
```

If there are no metal polygons in the data capture window, the denominator of this expression is 0. This does not satisfy the constraint, by definition.

### **Example 2**

The previous example can be altered to compute the ratio of metal to the area of the data capture window as follows:

```
metal_to_window {
  DFM ANALYZE metal WINDOW 100
  [AREA(metal) / AREA()] < z
}
```

If you do not specify a WINDOW parameter, the default data capture window is the extent of the layers read in for the run. Remember that Calibre only reads in the layers it actually needs for the run, which is not necessarily the entire layout database. DFM Analyze allows you to specify the WINDOW parameters without additionally specifying an RDB. This is different from other DFM operations.

### **Example 3**

You can organize the output by cell instead of by window. The following example shows this syntax:

```
metal_to_window {
  DFM ANALYZE metal
  [ AREA(metal) ] < z BY CELL
}
```

Again, specifying RDB output is optional.

# DFM Assert

Specification statement

```
DFM ASSERT {
    LAYERS '['[POLYGON] [EDGE] [ERROR] [GLOBAL] [LOCAL] [ORIGINAL]
    [NODAL]']' layer_name [layer_name ...] |
    NUMBER '['[POSITIVE] [NONNEGATIVE] [NONZERO] [INTEGER] [GRID]''
    expression [expression ...] |
    SCOPE {GLOBAL | LOCAL}
}
```

## Summary

Allows you to assert requirements on certain SVRF objects. If the specified requirements are not met, Calibre generates a compilation error.

## Parameters

- **LAYERS** '['*layer\_restrictions*'']' *layer\_name* [*layer\_name* ...]

A required keyword set that specifies one or more layer names along with any number of optional layer restrictions. If more than one layer restriction is specified, each layer name can be any of the specified types. If no layer restrictions are specified, all layer types are allowed (as if all of the *layer\_restrictions* keywords were specified). Possible choices for *layer\_restrictions* are:

- POLYGON — the layer must be a polygon layer (original or derived).
- EDGE — the layer must be a derived edge layer.
- ERROR — the layer must be a derived error layer.
- ORIGINAL — the layer must be an original layer.
- GLOBAL — the layer must be a global layer (derived outside of a rule check).
- LOCAL — the layer must be a local layer (derived inside a rule check).

- **NUMBER** '['*number\_restrictions*'']' *expression* [*expression* ...]

A required keyword set that specifies one or more numeric expressions along with any number of optional number restrictions. If no number restrictions are specified, any real number is allowed. Possible choices for *number\_restrictions* are:

- POSITIVE — the number must be positive (greater than zero).
- NONNEGATIVE — the number must be non-negative (greater than or equal to zero).
- NONZERO — the number must not be equal to zero.
- INTEGER — the number must be an integer.
- GRID — the number multiplied by the precision must be an integer.

- **SCOPE {GLOBAL | LOCAL}**

A required keyword set that controls the scope of the statement.

**GLOBAL** — the statement must appear outside of a rule check.

**LOCAL** — the statement must appear inside a rule check.

## Description

Allows you to assert requirements on layers and numeric expressions in a rule file. If the specified requirements are not met, Calibre generates a compilation error.

This statement is designed to support the creation of SVRF IP. One of the problems often encountered when creating IP using SVRF code involves error checking. If the inputs are incorrect or the containing file renders the IP incorrect, the errors are reported by whatever SVRF statement detects them. If the IP is encrypted, the errors are suppressed. If the IP is not encrypted, the errors can refer to internal objects (such as layers derived from input layers).

The DFM Assert statement can be used with regular or encrypted SVRF, SVRF DMACROS, and TVF procedures.

## Examples

### Example 1

The following statement asserts that ARG1 and ARG2 are layer names:

```
DFM ASSERT LAYERS ARG1 ARG2
```

Consider the above DFM Assert statement in the following SVRF context:

```
DMACRO CHECK ARG1 ARG2 {
    DFM ASSERT LAYERS ARG1 ARG2
    ...
}
LAYER M1 0
LAYER M2 1
m1m2 = M1 OR M2
m2_ext = EXT M1 < 0.3
CMACRO CHECK m1m2 m2_ext
CMACRO CHECK m1m2 M3
```

In the first CMACRO call, the macro arguments ARG1 and ARG2 resolve to the layer names m1m2 and m2\_ext, respectively. Since there are no restrictions on the types of layers they must be, this call compiles. In the second CMACRO call, the argument ARG2 resolves to the name M3. Assuming that there is no layer M3 defined in the rest of the rule file, the use of this macro triggers a RES2 error on layer name M3.

**Example 2**

The following statement asserts that ARG1 and ARG2 are polygon layers (original or derived), and have connectivity:

```
DFM ASSERT LAYERS [POLYGON NODAL] ARG1 ARG2
```

Consider the above DFM Assert statement in the following SVRF context:

```
DMACRO CHECK ARG1 ARG2 {  
    DFM ASSERT LAYERS [POLYGON NODAL] ARG1 ARG2  
    ...  
}  
LAYER M1 0  
LAYER M2 1  
CONNECT M1 M2  
m1m2 = M1 OR M2  
CMACRO CHECK m1m2 M2
```

The macro argument ARG1 resolves to the layer name m1m2. This is a polygon layer; however, it cannot have connectivity (since it is created by an OR operation and is not a Connect layer itself). Therefore, the use of this macro triggers an EXT3 error on layer m1m2.

Now, consider the above DFM Assert statement in the following SVRF context:

```
DMACRO CHECK ARG1 ARG2 {  
    DFM ASSERT LAYERS [POLYGON NODAL] ARG1 ARG2  
    ...  
}  
LAYER M1 0  
LAYER M2 1  
CONNECT M1 M2  
m1_length = LENGTH M1 > 1  
CMACRO CHECK m1_length M2
```

The macro argument ARG1 resolves to the layer name M1\_LENGTH. This is an edge layer with connectivity. Therefore, the use of this macro will trigger a TYP5 compilation error on layer M1.

**Example 3**

The following statement must not appear inside a rule check:

```
DFM ASSERT SCOPE GLOBAL
```

Consider the above DFM Assert statement in the following SVRF context:

```
DMACRO CHECK ARG1 ARG2 {  
    DFM ASSERT SCOPE GLOBAL  
    ...  
}  
LAYER M1 0  
LAYER M2 1  
RES {  
    CMACRO CHECK M1 M2  
}
```

This macro is used inside the check RES, which results in a LOC3 error.

**Example 4**

The following statements assert that ARG1 is a polygon layer and ARG2 is a positive number:

```
DFM ASSERT LAYERS [POLYGON] ARG1
DFM ASSERT NUMBER [POSITIVE] ARG2
```

Consider the above DFM Assert statements in the following SVRF context:

```
DMACRO CHECK ARG1 ARG2 {
    DFM ASSERT LAYERS [POLYGON] ARG1
    DFM ASSERT NUMBER [POSITIVE] ARG2
    ...
}
LAYER M1 0
LAYER M2 1
VARIABLE X 2
CMACRO CHECK M1 X-5
```

The macro argument ARG1 resolves to the layer name M1, which is a polygon layer. The macro argument ARG2 resolves to the numeric expression “X-5”, which, after variable substitution, evaluates to -3. Therefore, the use of this macro triggers a NUM3 error.

**Example 5**

Consider the following TVF procedure, which is contained in a file named my\_proc.tvf:

```
#!tvf
namespace import tvf::*
proc my_proc {layers} {
    foreach {layer} $layers {
        VERBATIM "DFM ASSERT LAYERS \[POLYGON\] ${layer}"
        RULECHECK check_${layer} {
            SETLAYER ${layer}_int = INT ${layer} <= 0.2
            OUTPUT "DFM RDB ${layer}_int out.rdb CHECKNAME ${layer}_int"
        }
    }
}
```

The DFM Assert statement in this procedure asserts that its input layer must be a polygon layer. Consider this procedure in the following TVF context:

```
tvf::VERBATIM {
LAYER M1 0
LAYER M2 1
m1_edge = EXT [M1] < 0.3
}

source my_proc.tvf

my_proc {M1 M2}
# my_proc {m1_edge M2}
```

The first call to my\_proc will compile, because M1 and M2 are both polygon layers. However, the second (commented) call to my proc will generate a TYP5 error because m1\_edge is not a polygon layer.

# DFM CAF

Layer operation

## **DFM CAF** *layer [layer ...]*

```
{SHORT [POLYGONAL] [{EXCLUDE FAULTS BETWEEN layerA '&' layerB} ...] |
OPEN [{PORTSLAYER port_layer1 [GROUPED BY group_layer1] } [PORTSLAYER
port_layer2 GROUPED BY group_layer2] }]
R r1 [r2 ...] [D depth] [METRIC {SQUARE | OCTAGONAL}]
[INSIDE OF {EXTENT / x1 y1 x2 y2 | LAYER markerlayer1 [markerlayer2 ...]}]
[WINDOW {w | x y} | NUMWIN {n | nx ny}]
[RESULT CELLS cell_list_name [WITHOUT TOPCELL]]
[HIERARCHICAL]
```

Used only in Calibre YieldAnalyzer applications.

## Summary

Calculates a numeric critical area value on one or more input layers. CAF stands for Critical Area Function.

## Parameters

- ***layer* [layer ...]**

A required argument that specifies one or more polygon (original or derived) input layers. Multiple input layers are allowed only if there is no interaction between the layers; that is, none of the shapes on one input layer touch or overlap any of the shapes on another input layer.

- **SHORT** [POLYGONAL] [{EXCLUDE FAULTS BETWEEN *layerA* '&' *layerB*} ...] |
**OPEN** [{PORTSLAYER *port\_layer1* [GROUPED BY *group\_layer1*] } [PORTSLAYER
*port\_layer2* GROUPED BY *group\_layer2*] ]

A required keyword set that specifies the type of failure for which critical area is to be calculated, which must be one of the following:

- **SHORT** — calculates shorts between different nets on the input layer.
- **OPEN** — calculates opens for nets on the input layer.

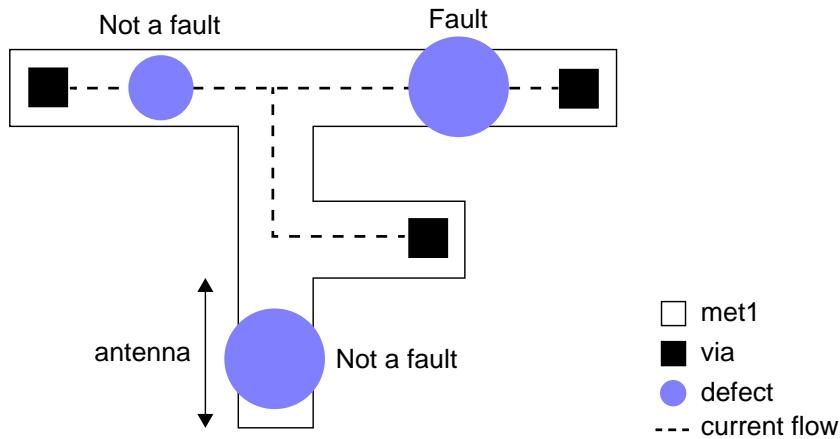
By default, only shorts between shapes belonging to different nets are considered faults. When the POLYGONAL keyword is specified, shorts between any two shapes are considered faults.

The EXCLUDE FAULTS BETWEEN keyword set, which may be specified multiple times, instructs the operation not to consider defects that short a *layerA* shape with a *layerB* shape. This keyword set is valid only when multiple input layers are specified. The arguments *layerA* and *layerB* can be the same layer, and can be specified in either order without affecting the output. The OCTAGONAL metric is not valid with this keyword set.

By default, any defect that covers two opposite edges of a shape is considered an open fault. The PORTSLAYER ... GROUPED BY keyword set modifies this behavior such that open

faults occur only where a defect breaks a path between two different port groups. Consider the example shown in [Figure 4-40](#).

**Figure 4-40. Port-aware Open CAA**



In this example, each via is considered a different port (a port is a location on a conducting shape from or toward which electrical current flows). Faults occur only where defects break the flow of current between the vias. Note that groups of polygonally-redundant vias are considered single ports.

The PORTSLAYER keyword set has three forms:

- PORTSLAYER *port\_layer1* — each shape on *port\_layer1* is a separate port group.
- PORTSLAYER *port\_layer1* GROUPED BY *group\_layer1* — each port group is defined by the set of all *port\_layer1* shapes that intersect with a *group\_layer1* shape.
- PORTSLAYER *port\_layer1* GROUPED BY *group\_layer1* PORTSLAYER *port\_layer2* GROUPED BY *group\_layer2* — defines two port groups and two group layers; each port group is defined by the set of all *port\_layer1* shapes that intersect with a corresponding *group\_layer1* shape. This form is useful for assigning port groups to via layers that lie both above and below a metal layer.
- **R *r1* [*r2 ...*]**  
A required keyword and argument set that specifies one or more defect radius values in user units.
- **D *depth***  
An optional keyword and argument pair that specifies the minimal overlap depth with a layout shape for a short or open to be considered a fault. The behavior of this keyword is identical to that of the D keyword in the [DFM Critical Area](#) operation.
- **METRIC {SQUARE | OCTAGONAL}**  
An optional keyword set that specifies the defect shape to use (the default is SQUARE).

- INSIDE OF {EXTENT /  $x1\ y1\ x2\ y2$  | LAYER *markerlayer1* [*markerlayer2* ...] }

An optional keyword and value set that defines the analysis region. By default, the analysis region is the layout extent. You can define a different analysis region by either specifying coordinates ( $x1\ y1\ x2\ y2$ ) or by specifying one or more marker layers with the LAYER keyword. If LAYER is used, the analysis region is the area inside of the specified marker layer(s).

If multiple marker layers are specified, the caf property, for each defect size, contains critical area values for each marker layer. For example, consider the following code:

```
// Calculates the CAF of M1 for open type defects inside areas
// covered by the shapes on the layers marker_layer_1 and
// marker_layer_2 for defect radius values 0.1, 0.2 and 0.3

M1_OPEN_CAF = DFM CAF M1 OPEN R 0.1 0.2 0.3 INSIDE OF LAYER
marker_layer_1 marker_layer_2
```

Assume that this code results in the following value for the caf property:

```
<0.1 156.13 236.03><0.2 498.65 311.91><0.3 604.84 488.30>
```

In table form, this data can be represented as:

Defect_size	marker_layer_1	marker_layer_2
0.1	156.13	236.03
0.2	498.65	311.91
0.3	604.84	488.30

- WINDOW { $w$  |  $x\ y$ } | NUMWIN { $n$  |  $nx\ ny$ }

An optional keyword and value set that controls partitioning of the output. By default, the output is calculated for the entire design. The WINDOW keyword allows you to partition the output into windows of the specified size. The NUMWIN keyword allows you to partition the output into the specified number of windows.

The windows in the top row and in the right-most column are truncated to align with the design extent. Windows that touch other windows on either the top or right sides are shrunk by 1 database unit in each direction, respectively, to prevent abutting shapes. Critical area values are calculated for these windows in their original, pre-shrunk form.

- RESULT CELLS *cell\_list\_name* [WITHOUT TOPCELL]

An optional keyword and value set that specifies a cell list. Each of the cells in the *cell\_list\_name* is processed as if it was a separate top cell. The actual top cell in the design is included implicitly unless WITHOUT TOPCELL is specified. The *cell\_list\_name* must appear in a Layout Preserve Cell List statement.

- HIERARCHICAL

An optional keyword that generates a property named self\_caf in addition to the caf property.

## Description

Calculates a numeric critical area value on one or more input layers. The output layer of this operation contains a vector DFM property named caf that reports critical area values for one or more defect sizes for the input layer(s). If multiple input layers are specified, the reported critical area value is the critical area for the union of the input layers. If marker layers are specified, a separate critical area value is reported for each marker layer. Critical area values are reported as if the design were flattened.

The [DFM GCA](#) operation can be used to generate a geometric representation of the calculated critical area values.

If the HIERARCHICAL keyword is specified, each cell contains, in addition to the caf property, a vector DFM property named self\_caf that reports critical area values in the context of each cell, excluding any placements.

A number of environment variables can be used to control the behavior of this operation:

- CALIBRE\_DFM\_OPEN\_GCA\_DEANGLING\_PRECISION — controls the precision (in user units) of the orthogonalization of critical area for OPEN faults. The default is 0.01. In general, smaller values can impact performance on non-rectilinear designs.
- CALIBRE\_DFM\_CAA\_MAX\_EDGES\_PER\_TILE — specifies the upper bound for the partition size as the average flat edge count per tile. The default value is 1,000,000. If this variable is set to a non-positive value, then every cell is processed as one tile. In order to enable the MT/MTflex parallelism of the DFM CAF operation this variable must be set to a reasonable value.
- CALIBRE\_DFM\_CAA\_MIN\_EDGES\_PER\_TILE — specifies the lower bound for the partition size as the average flat edge count per tile. The default value is 10,000. Low scalability of parallel processing can result when sparse layers are processed in fewer tiles. In these cases, the operation uses tiles of smaller size (but not less than the specified lower bound) so that all threads are utilized.

## Examples

### Example 1

```
// Calculates the total CAF of M1 for open type defects within
// the full design extent for defect radius values 0.1 0.2 0.3 0.4 0.5
M1_OPEN_CAF = DFM CAF M1 OPEN R 0.1 0.2 0.3 0.4 0.5
```

### Example 2

```
// Calculates the total CAF of M1 for short type defects within
// the full design extent for defect radius values 0.1 0.2 0.3 0.4 0.5
M1_SHORT_CAF = DFM CAF M1 SHORT R 0.1 0.2 0.3 0.4 0.5
```

### Example 3

```
// Power to signal short
// calculates critical area of defects causing shorts between a power
// and a signal net, but not power-to-power or signal-to-signal

// Assume that M1 can be separated into two layers M1power and
```

```
// M1signal containing shapes belonging to power and signal nets,  
// respectively. Then the critical area function for power-to-signal  
// short type faults is calculated by the DFM CAF operation  
  
M1_power2signal_CAF = DFM CAF M1power M1signal SHORT  
R 0.1 0.2 0.3 0.4 0.5  
EXCLUDE FAULTS BETWEEN M1power & M1power  
EXCLUDE FAULTS BETWEEN M1signal & M1signal
```

# DFM Classify

Layer operation

**DFM CLASSIFY** *layer\_name* [NOHIER] [ALL]  
**PROPERTY** *property\_name1* [{ABSOLUTE | RELATIVE} *tolerance1*]  
*property\_name2* [{ABSOLUTE | RELATIVE} *tolerance2*] ...]

## Summary

Groups geometries on a layer by one or more specified DFM properties and selects a representative geometry from each group.

## Parameters

- ***layer\_name***

A required argument that specifies the name of the input layer. This layer can be of any type (polygon, edge, or error layer).

- **NOHIER**

An optional keyword that when specified, outputs each representative geometry in the top cell along with placement information for each one.

When hierarchical Calibre executes the DFM Classify operation, geometries with equivalent property values are grouped together regardless of whether or not the geometries belong to the same cell. Representative geometries for each group can be chosen from any cell in which the group has at least one geometry. The selected representative geometry remains in the cell in which it exists on the input layer, unless NOHIER is specified. When NOHIER is specified, representative geometries (one for each group) are reported in the top cell. Each representative geometry contains, in addition to the Class DFM property that reports the group number, a property named Xform that reports placement information. When NOHIER is not specified, the representative geometries can be in any cell; if the cell is placed several times in the hierarchy, then the representative geometry will have several placements when considered flat.

- **ALL**

An optional keyword that when specified, instructs the operation to output all geometries on the input layer (so that the output layer is an exact copy of the input layer), and annotate each one with a DFM property named Class that contains the group number. Two DFM Classify operations, which differ only by the specification of the ALL option, run concurrently and produce consistent group numbers—if a geometry on the output layer of a DFM Classify operation without ALL specified has group number N, this geometry is a representative from the geometries annotated with the same group number N by the corresponding DFM Classify ALL operation.

- **PROPERTY** *property\_name1* [{ABSOLUTE | RELATIVE} *tolerance1*]  
*property\_name2* [{ABSOLUTE | RELATIVE} *tolerance2*] ...]

A required argument that specifies the names of one or more DFM property names associated with the input ***layer\_name***. The property names must refer to numeric-type

properties (other types of properties, such as vector properties, are not supported). For each *property\_name*, an optional *tolerance* value can be specified. If a *tolerance* value is not specified, it is assumed to be zero by default.

If a *tolerance* value is specified, you must specify either ABSOLUTE or RELATIVE. The definitions of “equal within tolerance” for these keywords are as follows:

- ABSOLUTE — if two geometries have property values A and B, then the geometries are in the same group only if:  $|A - B| \leq T$ , where T is the specified tolerance.
- RELATIVE — if two geometries have property values A and B, then the geometries are in the same group only if:  $|A - B| \leq |T * \min(A, B)|$ , where T is the specified tolerance and  $\min(A, B)$  is the minimum of A and B.

Note that the comparisons contain a small amount of leeway to account for round-off errors inherent in operations on floating-point numbers.

When values are compared within tolerance, it is possible that values A and B are considered equal within tolerance, values B and C are also equal, but values A and C are not equal (that is, the comparison is not transitive). For a given set of property values, the non-transitive comparison may or may not occur. For example, if the property values are 1.0, 1.1, 1.5, and 1.6, and the tolerance is 0.1, the non-transitive comparisons do not occur and the grouping of values is unambiguous. If, however, the property values are 1.0, 1.1, 1.2, and 1.3, then for the same tolerance 0.1, the grouping is ambiguous. For example, values 1.0 and 1.1 can be in the same group, or values 1.1 and 1.2 can be in the same group (in this case, the value 1.0 is not part of the group).

## Description

Groups geometries on a layer by one or more specified DFM properties and outputs a layer containing an arbitrary geometry from each group. Each representative geometry is annotated with a DFM property named Class that contains the group number (note that in general, group numbers do not start with 1). These properties cannot be referenced with the PROPERTY() function in DFM expressions, but can be accessed using Calibre YieldServer. Within each group, for each specified property name, the respective property values of all geometries are considered equal within the specified tolerance.

The classification done by the DFM Classify operation has the following attributes:

- Within each group, all property values for each property are equal within tolerance, whether or not any ambiguity exists in grouping of one or more properties.
- If the grouping is unambiguous for all properties, then between different groups, no two values are equal within tolerance. If the grouping by at least one property is ambiguous, then certain geometries in different groups may have property values that are equal within tolerance, and the input layer is not guaranteed to be partitioned into the smallest possible number of groups.

## Examples

### Example 1

Classify polygons on layer POLY by their area and select one representative polygon from each group; areas are considered equal if they are within 1% of each other.

```
POLY_A = DFM PROPERTY POLY [ A = AREA(POLY) ]
POLY_REP = DFM CLASSIFY POLY_A PROPERTY "A" RELATIVE 0.01
```

### Example 2

Classify polygons on layer GATE by the number of contacts in the regions on layer SD which touch each gate polygon, and by the area of the gate with a tolerance of 1%. Select one top-level placement for each representative gate polygon:

```
SD_N = DFM PROPERTY SD CONT [ N = COUNT(CONT) ]
GATE_NA = DFM PROPERTY GATE SD_N OVERLAP ABUT ALSO MULTI
          [ N = PROPERTY(SD_N, "N") ] [ A = AREA(GATE) ]
GATE_REP = DFM CLASSIFY GATE_NA PROPERTY "N" "A" RELATIVE 0.01 NOHIER
```

# DFM Compress

Layer operation

## DFM COMPRESS *input\_layer* FILL SPEC *spec\_name*

{**HLAYER** | **OUTPUT** *name*} [**PREFIX** *cell\_name\_prefix*] [**COLOR** *color\_name*]

Used only in Calibre YieldEnhancer applications.

### Parameters

- ***input\_layer***

A required argument that specifies a polygon layer. This layer contains the extent rectangles generated with a DFM Fill operation.

- **FILL SPEC *spec\_name***

A required keyword and argument set that specifies the name of a fill specification. This specification defines the OUTPUT, STEP and OFFSET parameters.

- **HLAYER**

A required keyword that specifies that the output layer is the hierarchy layer for a set of multi-output or single-output DFM Compress operations. The hierarchy layer contains the hierarchy (placement) information for all layers in the set. Exactly one hierarchy layer must be specified for each set of multi-output or single-output DFM Compress operations.

Additionally, the hierarchy layer must be included when writing one or more output layers using DFM RDB GDS or DFM RDB OADIS. The **HLAYER** keyword is not valid with **OUTPUT**.

- **OUTPUT *name***

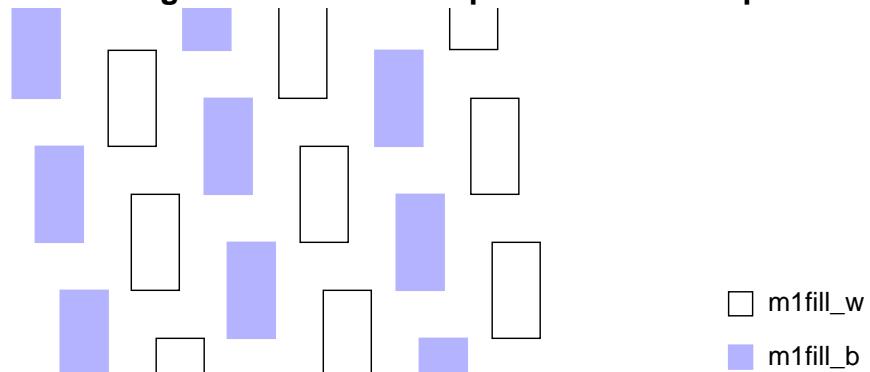
A required keyword and argument set that specifies a fill geometry to be output. The *name* must be defined with an OUTPUT keyword in the DFM Spec Fill statement for *spec\_name*.

- **PREFIX *cell\_name\_prefix***

An optional keyword and argument set that allows you to specify a prefix for cell names in a particular fill shape pattern. This argument is useful for avoiding naming conflicts that can occur when generating cell names. Multiple instances of *cell\_name\_prefix* must be unique for each DFM Spec Fill statement, and identical within a single DFM Spec Fill statement. The default begins with \_A, and increments alphabetically to \_Z. If more than 26 DFM Spec Fill statements are present in a rule file, a number is appended to the default prefix. Cell names are generated using a combination of the cell name prefix and an alphanumeric hash string. The hash string represents the specific combination of fill shapes, the number of repetitions, and orientation information.

- **COLOR *color\_name***

An optional keyword that enables splitting of fill output into two layers. The splitting is done such that shapes from the input layer are written to each output layer in an alternating pattern (refer to [Figure 4-41](#)). This is also known as double patterning.

**Figure 4-41. DFM Compress COLOR Output**

`m1fill_w` = DFM COMPRESS m1 FILL SPEC `m1fill_spec` COLOR "white"

`m1fill_b` = DFM COMPRESS m1 FILL SPEC `m1fill_spec` COLOR "blue"

## Description

Generates a compressed output layer by replacing repeating fill patterns (composed of up to 42 fill shapes) with equivalent cell placements. The output layer of this operation is intended to be used only with the DFM RDB operation with GDS or OASIS specified.

This operation is similar to the DFM Fill operation with COMPRESS specified. However, it offers the added advantage of allowing you to perform nmDRC checks prior to performing the compression. For example, in some cases it is desirable to generate a fill layer, run checks on the layer that may cause certain fill shapes to be removed, then apply compression to the remaining fill shapes.

Any INSIDE OF EXTENT, INSIDE OF `x1 y1 x2 y2`, or INSIDE OF LAYER options specified with `spec_name` are ignored. Fill shapes are placed directly into the extent rectangles for `input_layer`. In other words, the functionality of this operation is equivalent to running DFM Fill COMPRESS with the `input_layer` supplied to the INSIDE OF LAYER option for the specified `spec_name`. Similarly, any additional layer spacings and optimizer information must be applied when generating the envelopes with DFM Fill; these specifications are ignored by DFM Compress.

## Examples

This example shows a typical usage scenario for DFM Compress.

```
// DFM FILL layers
ezM2 = DFM FILL spec_bigd4 4ZM2
ezV1 = DFM FILL spec_bigd4 4ZV1
ezM1 = DFM FILL spec_bigd4 4ZM1
ezA4x = DFM FILL spec_bigd4 4ZA4 ENVELOPE 0.0

// DFM SPEC FILL SHAPE definitions
DFM SPEC FILL SHAPE BIG_1_0 POLYFILL 0 0 2.04 0 2.04 ... 0 2.04 0 2.04
DFM SPEC FILL SHAPE BIG_4_0 POLYFILL 0.48 0.24 0.6 0.24 ... 1.4 0.84 1.4
0.84 ...

```

## DFM Compress

---

```
DFM SPEC FILL SHAPE BIG_2_0 POLYFILL 0 0 0.12 0 ... 1.92 2.04 1.92 2.04
DFM SPEC FILL SHAPE BIG_3_0 POLYFILL 0.24 0.24 0.36 0.24 ... 1.2 0.84 1.2
    0.84 ..
DFM SPEC FILL SHAPE BIG_108_0 POLYFILL 0 0 2.04 0 2.04 2.04 0 2.04 0 2.04
2.04

// DFM SPEC FILL definition

DFM SPEC FILL spec_bigd4 z4_aof INSIDE OF LAYER z4_aof
    POLYFILL OUTPUT "4ZM4" "BIG_1_0"    OUTPUT "4ZV3" "BIG_4_0"
    OUTPUT "4ZM3" "BIG_2_0"    OUTPUT "4ZV2" "BIG_3_0"
    OUTPUT "4ZM2" "BIG_1_0"    OUTPUT "4ZV1" "BIG_4_0"
    OUTPUT "4ZM1" "BIG_2_0"    OUTPUT "4ZA4" "BIG_108_0"
    STEP 0.12 0.12 EFFORT 2 OFFSET 0.03 0.03 REPEAT 1

// nmDRC operations on ezM2, ezV1, ezM1

// construct layer ezA4s as the extents of "unedited" fill clusters
// (which will become cell placements). Layer ezA4s is derived from
// layer ezA4x. These layers are equal if the original fill is DRC
// clean.

dzM2 = DFM COMPRESS ezA4s FILL SPEC spec_bigd4 OUTPUT 4ZM2 PREFIX D4
dzV1 = DFM COMPRESS ezA4s FILL SPEC spec_bigd4 OUTPUT 4ZV1 PREFIX D4
dzM1 = DFM COMPRESS ezA4s FILL SPEC spec_bigd4 OUTPUT 4ZM1 PREFIX D4
dzH  = DFM COMPRESS ezA4s FILL SPEC spec_bigd4 HLAYER      PREFIX D4

BIG_DN { DFM RDB GDS "output.gds" dzM2 32 1 dzV1 51 1 dzM1 31 1 dzH 0 0 }
```

# DFM Connectivity Redundant

Layer operation

**DFM CONNECTIVITY [!]REDUNDANT *layer1* *layer2* [*layerN* ...]**

Used only in Calibre YieldAnalyzer applications.

## Parameters

- **!**  
Optional argument that when specified, instructs the operation to select non-redundant polygons instead of redundant polygons. If this argument is not specified, the operation selects redundant polygons.
- ***layer1***  
Required argument that specifies the layer on which redundant (or non-redundant) polygons are identified. This layer is typically a via layer, and must be a [Connect](#) layer.
- ***layer2* [*layerN* ...]**  
Required argument that specifies other layers involved in establishing connections to ***layer1***. All specified layers must be Connect layers.

## Description

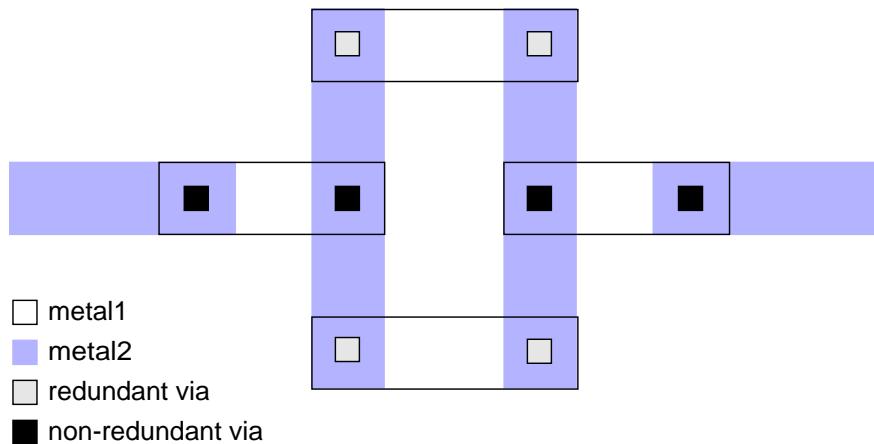
### Note



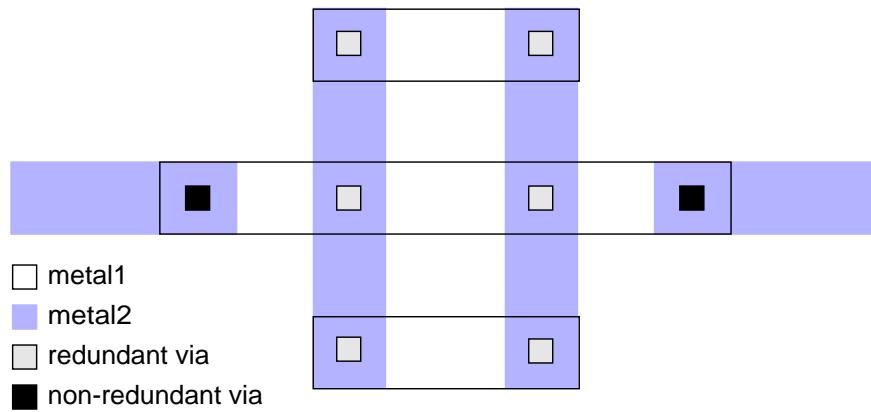
The DFM Connectivity Redundant operation is obsolete. It is recommended that you use the [DFM Redundant Vias](#) and [DFM Spec Via Redundancy](#) statements instead, which offer advantages in how hierarchy is treated.

Identifies redundant or non-redundant polygons on a layout and produces a derived polygon layer containing the identified polygons. A redundant polygon is defined as a polygon that may be removed without affecting the connectivity of the input layers. Only layers that are explicitly specified as input layers to this operation take part in determining redundancy.

In [Figure 4-42](#), all layers exist at the same level of hierarchy. The DFM Connectivity Redundant operation is used to identify redundant vias. If any one of the redundant vias indicated is removed, connectivity is unaffected.

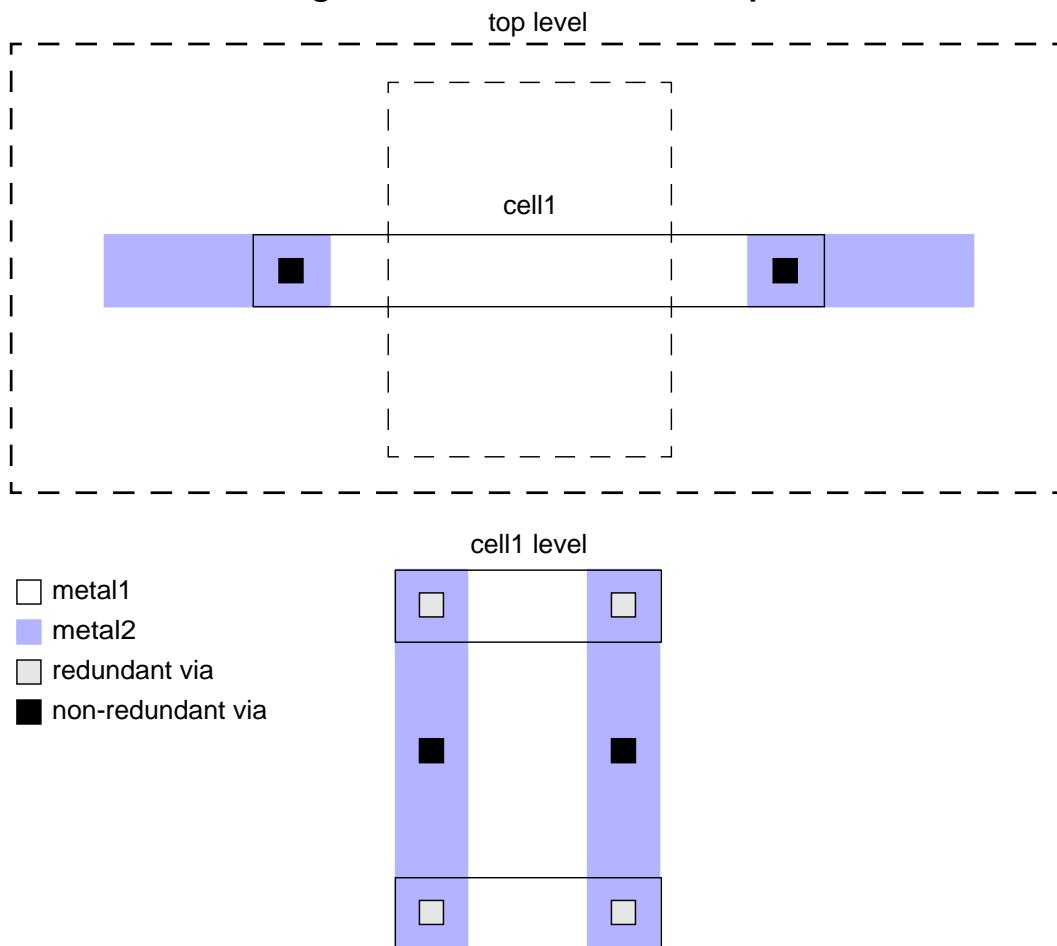
**Figure 4-42. Flat Example 1**

In [Figure 4-43](#), metal1 is extended through the center, and two additional vias are identified as redundant.

**Figure 4-43. Flat Example 2**

Redundancy is considered at each level of hierarchy without regard to the context in which placements of a particular cell may exist. In other words, no connections outside a placement of a cell can affect the redundancy of any polygons within that cell.

The example shown in [Figure 4-44](#) is identical to the example shown in [Figure 4-43](#), except that part of the design is drawn inside cell1. In this case, four vias at the cell1 level are identified as redundant. No vias are identified as redundant at the top level.

**Figure 4-44. Hierarchical Example**

The two center vias identified as redundant in [Figure 4-43](#) are not identified in this case because the metal1 polygon exists outside the placement of cell1.

This operation can also be used to identify redundant polygons on a metal layer, but may require additional SVRF code to function as desired. Consider the example shown in [Figure 4-45](#). If the operation is used as follows:

```
LAYER met1 1
LAYER via 2
LAYER met2 3
CONNECT met1 met2 BY via
met1_a = DFM CONNECTIVITY REDUNDANT met1 met2 via
```

no polygons in [Figure 4-45](#) are identified as redundant (refer to the results of met1\_a in the figure). Since DFM Connectivity Redundant operates on merged polygons, if the met1 polygon is removed, connectivity is broken.

However, if SVRF code is used to separate met1 into several polygons:

```
LAYER met1 1
LAYER via 2
LAYER met2 3
```

```

temp = (met1 AND met2) INTERACT via > 0
met1_new = met1 NOT temp
met2_new = met2 NOT temp

CONNECT met1_new temp
CONNECT met2_new temp

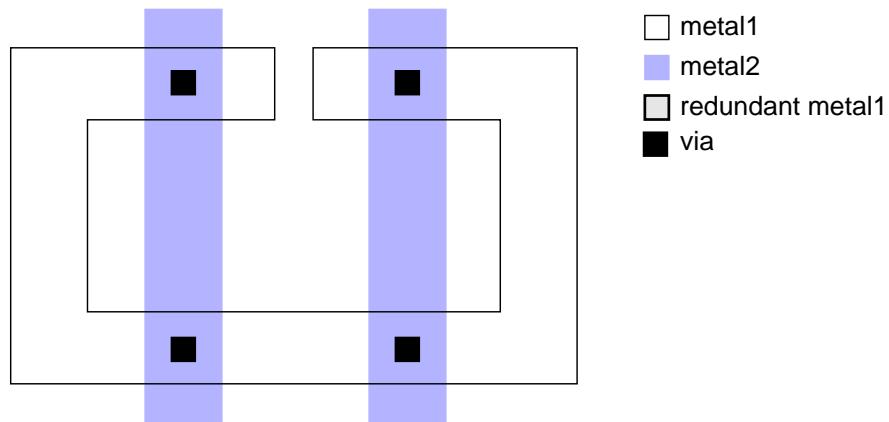
met1_b = DFM CONNECTIVITY REDUNDANT met1_new temp met2_new

```

the two shapes of met1 that appear on opposing sides are now identified as redundant (refer to the results of met1\_b in the figure).

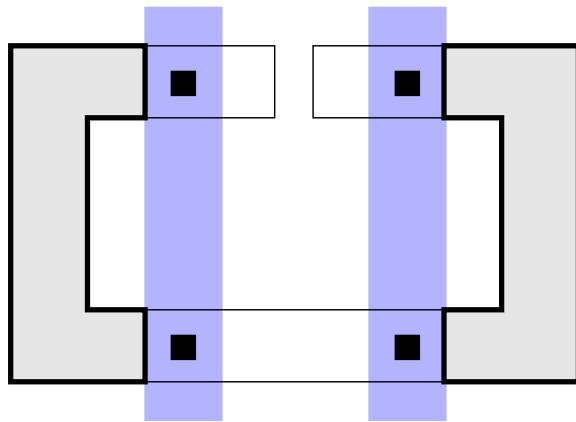
**Figure 4-45. Redundancy on Metal Layers**

Results of met1\_a:



- metal1
- metal2
- redundant metal1
- via

Results of met1\_b:



## Examples

### Example 1

```
// identifies redundant vias on layer via12  
vial2_r = DFM CONNECTIVITY REDUNDANT vial2 met1 met2 met3 via23
```

### Example 2

```
// identifies non-redundant vias on layer via12  
vial2_nr = DFM CONNECTIVITY !REDUNDANT vial2 met1 met2 met3 via23
```

# DFM Copy

Layer operation

**DFM COPY** *layer* [*layer* ...] [REGION [UNMERGED] | EDGE | CENTERLINE | TIE [CENTER] | CLUSTER [LAYERID] | UNMERGED] [CELL LIST *list\_name*]

## Parameters

- ***layer*** [*layer* ...]

A required original layer, derived polygon layer, derived edge layer, or derived error layer. If more than one ***layer*** is specified, all input layers are copied into a single output layer, merging for certain cases (refer to [Merging Behavior](#)). All input layers must be of the same type. Node numbers and DFM properties are copied to the output layer when a single input layer is specified and no secondary keywords are used. Node numbers and DFM properties are not preserved when multiple input layers are specified.

- REGION [UNMERGED]

Optional keyword that instructs the operation to convert error clusters on ***layer*** into polygons, and output a polygon layer. This keyword may be used only if ***layer*** is a derived error layer. Error clusters are converted as follows:

- Error clusters with two edges are converted by filling the space between the edges, such that the two edges become the sides of a polygon.
- Error clusters with more than two edges are converted into polygons using the convex hulls of the edges.
- Error clusters of one or more edges, where all edges are collinear, are expanded into rectangles of width approximately 4 database units.

Note that resulting polygons are merged within each cell, so polygons from different error clusters are merged if they touch or overlap. If this is not the desired behavior, the UNMERGED keyword can be specified to produce an output layer containing unmerged polygons. The output layer is annotated with DFM properties copied from the input layer — every output polygon carries the property values from the input edge cluster used to construct the polygon. The output layer of a DFM Copy REGION UNMERGED operation cannot be used in SVRF operations that expect a merged input polygon layer. Only this operation, [DFM Property Merge](#) and [DFM RDB](#) allow unmerged polygon layers as input.

- EDGE

Optional keyword that instructs the operation to convert error clusters on ***layer*** into edges, and output an edge layer. This keyword may be used only if ***layer*** is a derived error layer. Each edge of a cluster is copied to the output layer as a separate edge. Note that edges on the output layer are merged within each cell, so edges from different error clusters are merged if they are colinear and share at least one point.

- **CENTERLINE**

Optional keyword that instructs the operation to convert error clusters on *layer* that consist of two edges into a single edge, and output an edge layer that is the centerline between the error cluster. The resultant edge faces an arbitrary direction. This keyword may be used only if *layer* is a derived error layer. Only error clusters that consist of two edges are converted.

- **TIE [CENTER]**

Optional keyword that instructs the operation to convert error clusters on *layer* that consist of two edges into a single edge, and output an edge layer that is a tie connecting the two edges in the cluster. By default, the tie represents the shortest distance between the two cluster edges. If the cluster edges are parallel and project onto each other, the tie is placed in the center of the projection region. The optional CENTER keyword places the tie at the center points of the two edges in the cluster.

- **CLUSTER [LAYERID]**

Optional keyword set that combines one or more input edge layers into a single output derived error layer composed of single-edge clusters. Edges are not merged before the conversion, so the single-edge clusters can contain coincident edges. The optional LAYERID keyword instructs the operation to assign a numeric DFM property named LID to each output edge that indicates the input layer that the edge is derived from (for example, a property value of 1 means the edge is derived from the first input layer, 2 means it is derived from the second input layer, and so on). This keyword set is valid only when all input layers are derived edge layers. DFM properties on the input layers are not preserved when using this keyword set.

- **UNMERGED**

Optional keyword that must be specified if the input layer is an unmerged polygon layer. Unmerged polygon layers can be created with [DFM Text](#), [DFM Read](#), [DFM Expand Edge UNMERGED](#), or [DFM Copy REGION UNMERGED](#).

- **CELL LIST *list\_name***

Optional keyword that controls which geometries from *layer* are copied to the output layer. If this keyword is specified, only geometries in the cell list specified by *list\_name* are copied to the output layer (cell lists are created with the [Layout Cell List](#) statement). If this keyword is not specified, all geometries from *layer* are copied to the output layer. This keyword is applied to all input layers if more than one is specified.

## Description

Copies layers of any type to a new layer, and allows selection of specific cells to copy.

If more than one polygon *layer* is specified, the output layer is merged and contains copies of all polygons from all input layers. Therefore, the operation:

```
P = DFM COPY A B C D
```

is equivalent to:

```
P = (A OR B) OR (C OR D)
```

If more than one edge *layer* is specified, the output layer is merged and contains copies of all edges from all input layers. Therefore, the operation:

```
E = DFM COPY A B C D
```

is equivalent to:

```
E = ( A OR EDGE B ) OR EDGE ( C OR EDGE D )
```

If more than one error *layer* is specified, the output layer contains copies of all edge clusters from all input layers. If REGION is specified, all edge clusters are converted to polygons and resulting polygons from all input layers are copied into the output layer and merged, as if by the OR operation. If EDGE or CENTERLINE are specified, all edge clusters are converted to edges and resulting edges from all input layers are copied into the output layer and merged, as if by the OR EDGE operation.

When REGION, EDGE or CENTERLINE are specified, node numbers and DFM properties are not preserved. These keywords may be used together with the CELL LIST keyword.

## Merging Behavior

If multiple merged polygon input layers that abut or overlap are specified, or multiple colinear edges or edge clusters are specified, DFM Copy merges them upon output. This operation also accepts unmerged layers as input. [Table 4-7](#) describes how this unmerged data is processed.

**Table 4-7. DFM Copy Merging Behavior for Unmerged Input**

Unmerged Input	Output
Single unmerged edge layers.	Not merged within cells and across cells.
Multiple unmerged edge layers.	Merged within cells and across cells (except when CLUSTER is specified).
Single and multiple unmerged error layers.	Not merged within cells and across cells.
Single and multiple unmerged polygon layers.	Not merged within cells and across cells.

## Examples

### Example 1

Copies the error layer D to a new layer D1:

```
D = EXTERNAL MET <= 0.1
D1 = DFM COPY D
```

### Example 2

Copies all geometries in cells that start with “A” from layer MET to layer MET\_A:

```
LAYOUT CELL LIST A "A*"
MET_A = DFM COPY MET CELL LIST A
```

**Example 3**

Copies all geometries in cells that do not start with “A” from layer MET to layer MET\_NOTA:

```
LAYOUT CELL LIST NOTA "*" "!A*"  
MET_NOTA = DFM COPY MET CELL LIST NOTA
```

Another way to accomplish the same task using a regular expression:

```
LAYOUT CELL LIST NOTA "^[^A]"  
MET_NOTA = DFM COPY MET CELL LIST NOTA
```

**Example 4**

Converts an error layer to a polygon layer:

```
MET_S = EXT MET <= 0.1  
MET_SR = DFM COPY MET_S REGION
```

**Example 5**

Copies polygon layers A and B into a single, merged output layer:

```
A_B = DFM COPY A B
```

**Example 6**

Converts two-edge clusters to single edges. The edges are located on the centerline between the two edges in the cluster and face an arbitrary direction:

```
MET_S = EXT MET <= 0.1  
MET_SCL = DFM COPY MET_S CENTERLINE
```

**Example 7**

Converts derived edge layers MET1\_EDGE and MET2\_EDGE into a single derived error layer composed of single-edge clusters.

```
MET_ERROR = DFM COPY MET1_EDGE MET2_EDGE CLUSTER
```

# DFM Create Layer

Layer operation

**DFM CREATE LAYER POLYGONS** [NODAL] [ORDERED] '['  
*polygon\_specification* [*polygon\_specification* ...]  
 '']

Where the syntax for *polygon\_specification* is:

{*vertex\_count* *x1 y1 x2 y2* [ ... *xn yn*] [ NET *net\_num*] [CELL *cell\_name*] }

## Summary

A layer operation that creates a new layer containing polygons as specified.

### Note



This command can only be used inside the dfm::new\_layer command in a Calibre YieldServer script.

## Parameters

- **NODAL**

An optional argument instructing the Calibre hierarchical engine to assign node numbers to the polygons on the newly created layer. If NODAL is specified, a *net\_num* must be supplied for each polygon and these *net\_nums* must be numbers for nets that exist in the specified cell. If it is not specified, specifying a *net\_num* will result in an error.

- **ORDERED**

An optional argument that instructs the Calibre hierarchical engine to assign fixed polygon numbers to each polygon specified with *polygon\_specification*. This argument must be specified if properties are added to the new layer.

- [ ]

Syntactical elements required to enclose the *polygon\_specification* (and any additional instances). Square brackets that are true syntactical elements appear in the syntax description as '[' and ']'.

- ***polygon\_specification* [*polygon\_specification* ...]**

A required argument set defining a polygon to be created on the new layer. You must include at least one *polygon\_specification*. The syntax is as follows:

{*vertex\_count* *x1 y1 x2 y2* [ ... *xn yn*] [ NET *net\_num*] [CELL *cell\_name*] }

- ***vertex\_count***

A required argument specifying the number of vertices for a polygon to be created on the new layer.

- ***x1 y1 x2 y2 [ ... xn yn ]***

A required argument set defining the vertices of the polygon. The vertices are checked to ensure that they are not degenerate or unmergeable. This is more flexible than [Polygon](#) or [Layout Polygon](#), which require simple polygons.

- **NET *net\_num***

A keyword and argument assigning a net number to the polygon. This keyword is required when NODAL is specified and forbidden if it is not specified.

- **CELL *cell\_name***

An optional keyword and argument defining the cell that is the parent of this polygon. If this keyword is not specified, the polygon is added to the top cell.

## Description

Creates a new layer containing the polygons defined by the *polygon\_specification*.

During execution, the polygons are merged to create the output layer. It is an error if NODAL is specified and the number of polygons in the output layer is different from the number of polygons in the command input.

## Examples

```
// Creates three polygons, each with 4 verticies on the new
// layer. For each polygon, verticies are specified as
// x1 y1 x2 y2 x3 y3 x4 y4

DFM CREATE LAYER POLYGONS [
 4 0.0 25.0 20.0 25.0 20.0 50.0 0.0 50.0
 4 0.0 50.0 20.0 50.0 20.0 75.0 0.0 75.0
 4 0.0 75.0 20.0 75.0 20.0 100.0 0.0 100.0
]
```

## DFM Critical Area

Layer operation

### **DFM CRITICAL AREA** *layer1*

```
{SHORT [EXCLUDE EDGE exclude_layer] [POLYGONAL] | OPENWIRE |
OPENUWIRE | OPENVIA [enclosure_layer]}
[D depth] [SPATIAL SAMPLE spec_name]
S defect_radius [S defect_radius ...] [OUTPUT]
[METRIC SQUARE] [REGION OCTAGONAL]
```

Used only in Calibre YieldAnalyzer applications.

### Summary

Layer operation that is used to identify yield-limiting conditions. It performs single-layer checks and highlights areas in your layout that are susceptible to producing open or short circuits in the presence of random particles of a given radius.

### Parameters

- ***layer1***  
A required original or derived polygon layer. This is an interconnect layer.
- **SHORT | OPENWIRE | OPENUWIRE | OPENVIA** [*enclosure\_layer*]  
A required keyword set specifying the type of failure for which critical area is to be calculated. Must be one of the following:
  - **SHORT** — Selects shorts between two different nets on the interconnect layer (*layer1*).
  - **OPENWIRE** — Selects opens on the interconnect layer, and defects covering line-end portions of the interconnect layer. Refer to [Example 3, Method A: Use OPENWIRE](#) for an example of how to use this option.
  - **OPENUWIRE** — Selects opens on the interconnect layer. Because this type of analysis recognizes the line-end portions of the interconnect (based on edge and corner conditions), it does not report defects that cover line-ends, which do not block current flow. To ensure that defects blocking contacts or vias are caught, this option should always be used together with an operation using **OPENVIA**, as illustrated in [Example 3, Method B: Use OPENUWIRE plus OPENVIA](#).
  - **OPENVIA** [*enclosure\_layer*] — Selects opens on via portions of the interconnect layer. This keyword may be used without an operation using **OPENUWIRE** to catch defects blocking vias only, or in conjunction with an operation using **OPENUWIRE** to catch defects blocking metal regions above or below vias in addition to the vias themselves.

When evaluating a layout for this type of failure, you can include the optional

keyword, *enclosure\_layer*, to specify a derived polygon layer that identifies via arrays.

Derivation of this layer is typically something such as:

```
vial_enc = (M1 AND M2) ENCLOSE vial > 1
```

where *vial\_enc* encloses the *vial* arrays. If not specified, all vias are assumed to be non-redundant.

- EXCLUDE EDGE *exclude\_layer*

Optional keyword and polygon layer used only with **SHORT** calculations. This keyword causes edges from *layer1* to be excluded from the calculation when *layer1* edges are coincident with *exclude\_layer* edges. This option is often used to suppress false critical area results on gate regions when extracting **SHORT** critical area on the active layer.

- POLYGONAL

Optional keyword that when specified, enables hyperscaling for **SHORT** calculations. If POLYGONAL is not specified in a hyperscaled run, the operation runs on HDB0 only and produces identical results to a non-hyperscaled run.

Note that using this keyword may cause the operation to produce slightly different results. By default, DFM Critical Area is a nodal operation, that is, it uses hierarchical connectivity information present on *layer1*. The POLYGONAL keyword allows **SHORT** calculations to be made without hierarchical connectivity information. In other words, when POLYGONAL is specified, critical area may be generated between polygons that are connected elsewhere in the hierarchy. “U” shapes, notches, or other shapes from single polygons within a cell do not generate critical area, regardless of whether or not this keyword is specified.

Using this keyword produces identical results between hyperscaled and non-hyperscaled runs.

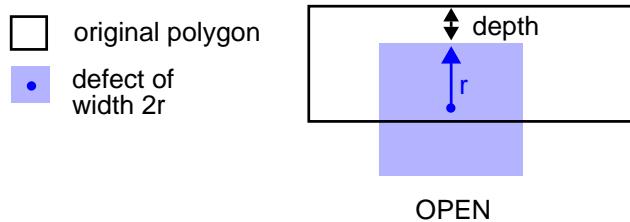
No additional keywords are required for using hyperscaling with DFM Critical Area with the **OPENWIRE**, **OPENUWIRE** and **OPENVIA** keywords. In addition, results for these types of checks match results generated without hyperscaling.

- D *depth*

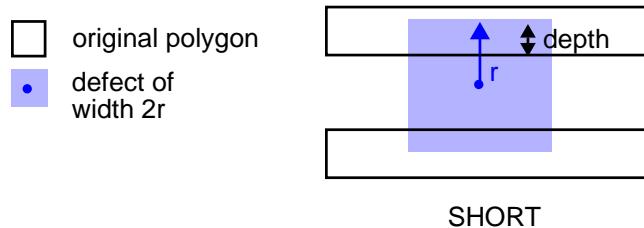
Optional secondary keyword and floating-point numeric value that specifies a distance between a defect boundary and the edge of an input layer polygon. The operation interprets this distance differently depending on the type of failure being investigated:

- For **OPEN** cases, this is the maximum amount of material you believe must remain if current is to flow properly. It defines the distance between the exterior-facing edge

of a defect and the interior-facing edge of a wire polygon. As the *depth* increases for a given S radius, the critical area increases.



- For **SHORT** cases, this is the minimum amount of overlap you believe causes a short. It is the distance between the inside-facing edge of a defect and the edge of an input polygon (or wire), which must be this size or larger to make a connection. As D decreases for a given S radius, the critical area increases.



- The *depth* must be a non-negative real number expressed in user units. It can be supplied as a numeric value or a variable. If you do not specify the D keyword, the defect *depth* is assumed to be 0.

This keyword set may appear once per rule check. You must create separate layer derivations for each D value in which you are interested.

- **SPATIAL SAMPLE** *spec\_name*

Optional keyword and value that specifies the name of a specification defined by DFM Spec Spatial Sample. This keyword cannot be used with **OPENVIA** or **OPENUWIRE**.

- **S** *defect\_radius* [S *defect\_radius* ...]

Required keyword and floating-point value that specifies the radius (or semi-diameter) of a circle. The *defect\_radius* must be a non-negative real number expressed in user units. It can be supplied as a numeric value or a variable. The actual defect (by default) is modeled by a square with the circle inscribed. This parameter set can be specified multiple times.

- **OUTPUT**

Optional keyword that specifies which *defect\_radius* parameter to use for determining the critical area output. This keyword is positional. It must immediately follow the *defect\_radius* parameter of interest.

This keyword may only be specified once. It is only needed if more than one S keyword set appears in the rule. If this keyword is omitted, the largest *defect\_radius* in the rule is used, which is inclusive of all radii that are less than or equal to the largest.

- METRIC SQUARE

Optional keyword that instructs the tool to use the square metric when determining edges within the defect radius (by default, the Euclidean metric is used), and a square defect model for critical area region generation. When this keyword is specified, Calibre generates critical area using a true representation of a square defect.

- REGION OCTAGONAL

Optional keyword that instructs the tool to use the Euclidean metric when determining edges within the defect radius (the default), and equilateral octagons instead of squares for critical area region generation.

## Description

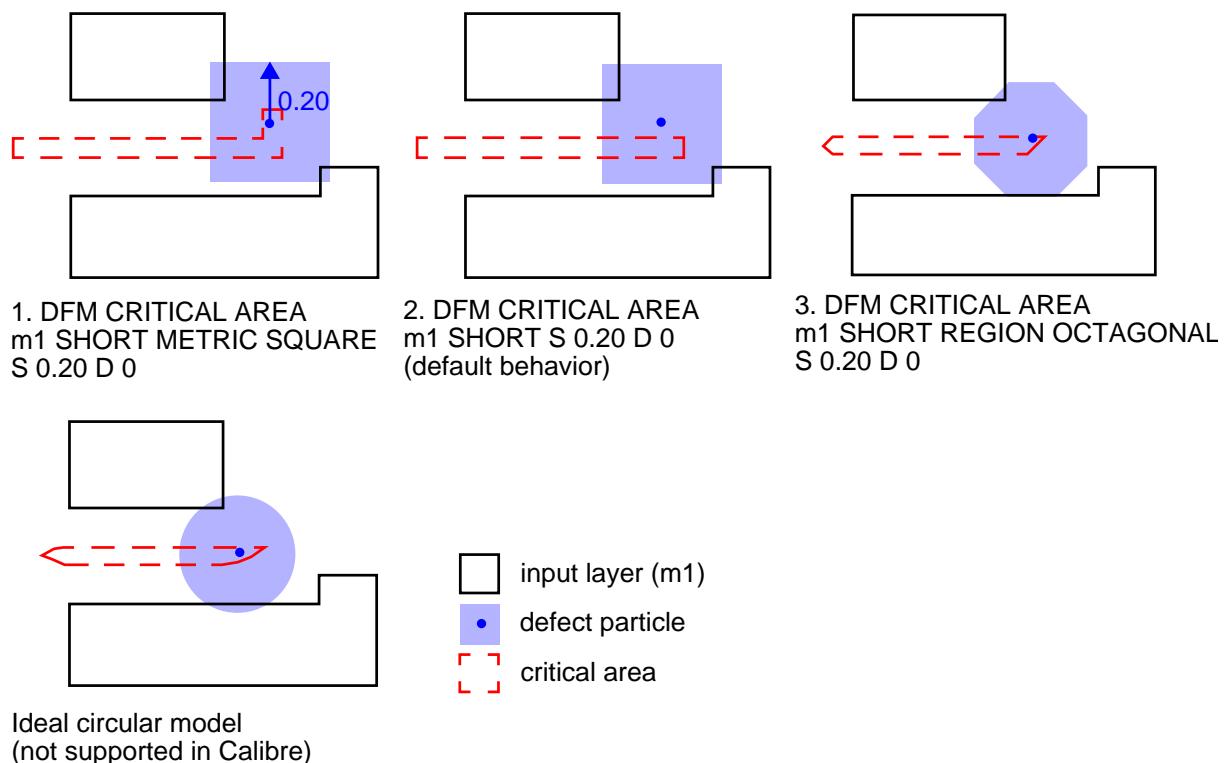
This operation is used for finding areas (called critical areas) of a layout where a defect of a given radius is likely to cause either an open or a short circuit. The **SHORT**, **OPENWIRE**, **OPENUWIRE**, and **OPENVIA** keywords specify the type of failure, namely, a short or open circuit.

The shape of the defect has an effect on the size of the critical area. It is generally accepted that a circular defect is the most accurate model of the irregularly shaped defects encountered in practice. Because of runtime constraints, the DFM Critical Area operation assumes (by default) a square defect of width  $2r$ . This results in a slightly larger critical area than what would result from using a circular defect of radius  $r$ . The calculation of the critical area for a square defect is more efficient, because the result for orthogonal input is an orthogonal critical area, which can be stored and analyzed more easily. Calibre can also calculate critical area using octagonal defect shapes.

The METRIC SQUARE and REGION OCTAGONAL keywords allow you to change the measurement method Calibre uses to determine edges within the defect radius and the shape of the defect model used. These options can be used to change critical area accuracy with respect to runtime. Refer to [Table 4-8](#) and [Figure 4-46](#).

**Table 4-8. Critical Area Generation Methods**

Keyword	Measurement Method	Defect Model Shape	Accuracy	Computation Speed
METRIC SQUARE	square	square	lowest	fast
None (default)	Euclidean	square	moderate	fast
REGION OCTAGONAL	Euclidean	octagon	highest	moderate

**Figure 4-46. Critical Area Regions for SHORT**

## Concurrency

All DFM Critical Area operations with the same input layer, failure type, D parameter, and S parameters are performed concurrently.

Suppose you want to derive the critical area regions where the ranges of defects are as follows:

- 0 - 0.01
- 0.01 - 0.015
- 0.015 - 0.02

Some defects are much more common than others, so you want the critical areas for each range of defect sizes reported on different layers. Also, suppose that the maximum overlap of a defect with a metal polygon is 0.005, where the defect could cause a short.

How you write your layer derivations will have a major impact on run-time.

These layer derivations would be processed separately:

```
X = DFM CRITICAL AREA m1 SHORT S 0.01 OUTPUT D 0.005
Y = DFM CRITICAL AREA m1 SHORT S 0.015 OUTPUT D 0.005
Z = DFM CRITICAL AREA m1 SHORT S 0.02 OUTPUT D 0.005
```

However you can cut processing time by two thirds if you write the layer derivations in such a way as to be processed concurrently:

```
X = DFM CRITICAL AREA m1 SHORT S 0.01 OUTPUT S 0.015 S 0.02 D 0.005
Y = DFM CRITICAL AREA m1 SHORT S 0.01 S 0.015 OUTPUT S 0.02 D 0.005
Z = DFM CRITICAL AREA m1 SHORT S 0.01 S 0.015 S 0.02 OUTPUT D 0.005
```

The location of the OUTPUT keyword dictates which defect radius is used for the output data.

It is important to understand that when you are using concurrent operations, the tool calculates the critical area for each of the sizes listed, regardless of whether or not the results are reported. Continuing with this example, if you comment out the third concurrent operation, the tool calculates the critical area for all three sizes and only outputs the first two.

```
X = DFM CRITICAL AREA m1 SHORT S 0.01 OUTPUT S 0.015 S 0.02 D 0.005
Y = DFM CRITICAL AREA m1 SHORT S 0.01 S 0.015 OUTPUT S 0.02 D 0.005
//Z = DFM CRITICAL AREA m1 SHORT S 0.01 S 0.015 S 0.02 OUTPUT D 0.005
```

In other words, you get the optimal performance when concurrent operations investigate only the sizes that you are interested it. In the case of our example, if we are interested only in finding shorts induced by particles in the 0 to 0.01 and 0.01 to 0.015 ranges, we get the best performance when the operation does not include the value S 0.02.

```
X = DFM CRITICAL AREA m1 SHORT S 0.01 OUTPUT S 0.015 D 0.005
Y = DFM CRITICAL AREA m1 SHORT S 0.01 S 0.015 OUTPUT D 0.005
```

## Examples

### Example1

Finding single-layer shorts between two electrical nodes.

Assume:

You want to identify the critical areas on layer m1.

You are interested in shorts between wires on different nets.

You are concerned about three specific particle sizes: 0.5, 1.0, and 2.0.

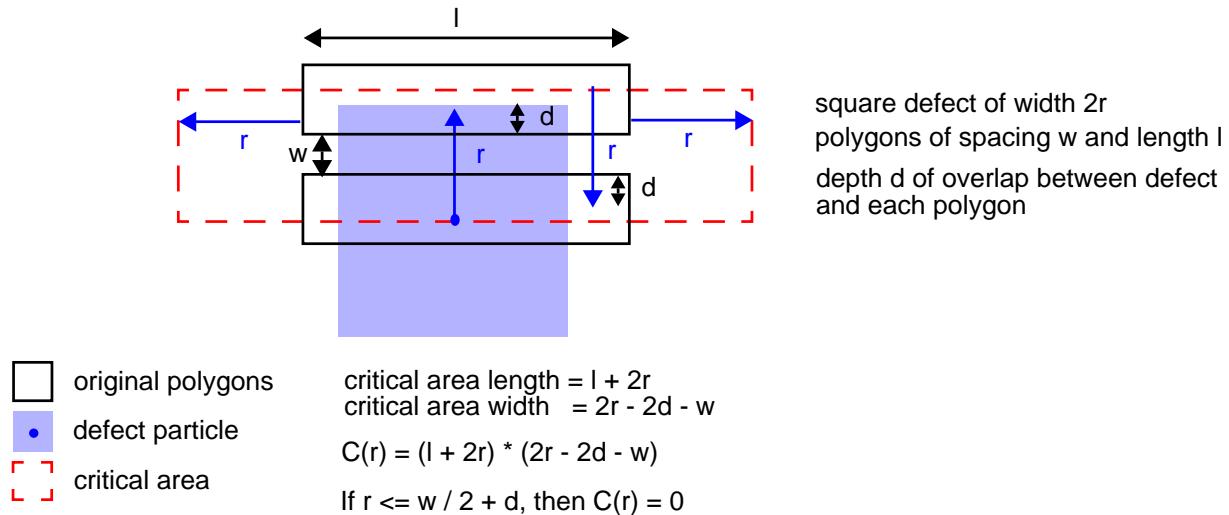
You want the critical area for each particle size saved as a different output layer.

You know a particle must overlap a polygon by at least 0.25 microns to cause a problem.

The set of rules for performing a critical area check for this type of short circuit would be as follows:

```
m1_short1 = DFM CRITICAL AREA m1 SHORT S 0.5 OUTPUT S 1.0 S 2.0 D 0.25
m1_short2 = DFM CRITICAL AREA m1 SHORT S 0.5 S 1.0 OUTPUT S 2.0 D 0.25
m1_short3 = DFM CRITICAL AREA m1 SHORT S 0.5 S 1.0 S 2.0 OUTPUT D 0.25
```

[Figure 4-47](#) shows how the operation identifies critical areas for wire regions on a single layer.

**Figure 4-47. Single-Layer Short Critical Area Model**

In Figure 4-47, the dimension  $d$  is the distance between an interior-facing edge of a wire and the interior-facing edge of a defect. This distance is the minimum overlap of a wire polygon by a defect, and this overlap is called *depth*.

In terms of the SVRF EXTERNAL operation, if the edges of the interconnect polygons satisfy this operation:

**EXTERNAL layer <  $2 * (r - d)$**

then they lie within a critical area region. The critical area region is extended by the defect radius length from each end of the wire.

### Example 2

Finding open defects caused by compromised vias.

Assume:

You want to identify the critical areas on layer via1.

You are concerned about defects covering vias in such a way as to inhibit current flow.

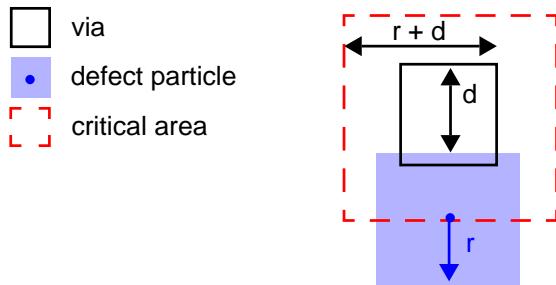
You are concerned about three specific particle sizes: 0.5, 1.0, and 2.0.

You want the critical area for each particle size saved as a different output layer.

You know a particle must overlap a via by at least 0.25 microns to cause a problem.

The set of rules for performing a critical area check for this type of defect would be as follows:

```
vial_open1 = DFM CRITICAL AREA m1 OPENVIA S 0.5 OUTPUT S 1.0 S 2.0 D 0.25
vial_open2 = DFM CRITICAL AREA m1 OPENVIA S 0.5 S 1.0 OUTPUT S 2.0 D 0.25
vial_open3 = DFM CRITICAL AREA m1 OPENVIA S 0.5 S 1.0 S 2.0 OUTPUT D 0.25
```

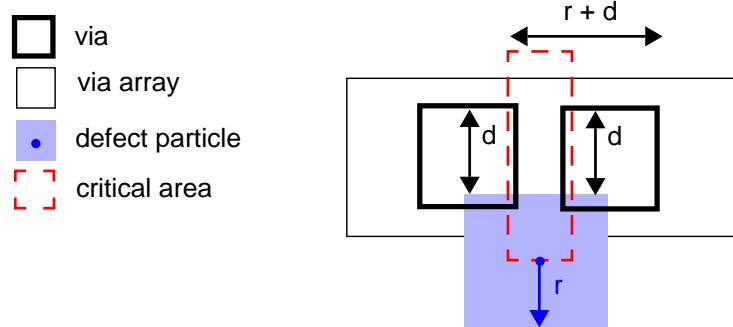
**Figure 4-48. Isolated Open Via Critical Area Model**

In [Figure 4-48](#), the dimension  $d$  is the distance between an interior-facing edge of an isolated via and the exterior-facing edge of a defect. This distance is the maximum width of a via left remaining by a defect.

For via arrays, DFM Critical Area you distinguish between via types by supplying a marker layer that encloses banked vias. The derivation of this marker layer might be something like this:

```
bank_vial = (M1 and M2) ENCLOSE vial > 1
```

Each via array edge is expanded, as for isolated vias, and the intersection of these expanded edges, which is sufficient to compromise the entire via array, determines the critical area region. [Figure 4-49](#) shows an example.

**Figure 4-49. Open Via Critical Area Mode for Via Arrays**

Performing a critical area check for a via layer containing via arrays might be as follows:

```
vial_open1 = DFM CRITICAL AREA via1 OPENVIA bank_vial
            S 0.5 OUTPUT S 1.0 S 2.0 D 0.25
vial_open2 = DFM CRITICAL AREA via1 OPENVIA bank_vial
            S 0.5 S 1.0 OUTPUT S 2.0 D 0.25
vial_open3 = DFM CRITICAL AREA via1 OPENVIA bank_vial
            S 0.5 S 1.0 S 2.0 OUTPUT D 0.25
```

**Example 3**

Finding single-layer wire open defects.

Assume:

You want to identify the critical areas on layer m1.

You are concerned about defects breaking wires on any net.

You are concerned about three specific particle sizes: 0.5, 1.0, and 2.0.

You want the critical area for each particle size saved as a different output layer.

You know that even a particle landing 0.25 microns from the edge of a wire can cause an electrical OPEN.

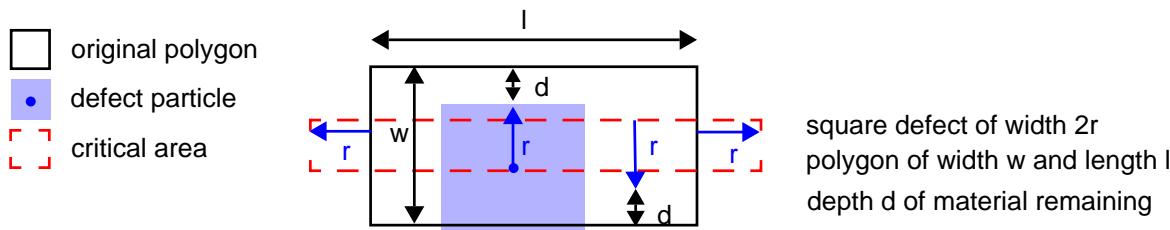
You can perform this check using either of two methods:

**Method A: Use OPENWIRE**

```
m1_open1 = DFM CRITICAL AREA m1 OPENWIRE S 0.5 OUTPUT S 1.0 S 2.0 D 0.25
m1_open2 = DFM CRITICAL AREA m1 OPENWIRE S 0.5 S 1.0 OUTPUT S 2.0 D 0.25
m1_open3 = DFM CRITICAL AREA m1 OPENWIRE S 0.5 S 1.0 S 2.0 OUTPUT D 0.25
```

A portion of the width of the original polygon can remain intact, but this is still modeled as an open. [Figure 4-50](#) shows how this is modeled for wire regions.

**Figure 4-50. Single-Layer Open Critical Area Model**



$$\begin{aligned} \text{critical area length} &= l + 2r \\ \text{critical area width} &= 2r + 2d - w \end{aligned}$$

$$C(r) = (l + 2r) * (2r + 2d - w)$$

$$\text{If } r \leq w / 2 - d, \text{ then } C(r) = 0$$

In [Figure 4-50](#), the dimension **d** is the distance between an interior-facing edge of a wire and the exterior-facing edge of a defect. This distance is the maximum width of a polygon left remaining by a defect.

**Method B: Use OPENWIRE plus OPENVIA**

DFM Critical Area uses a different heuristic for general interconnect layer regions. The OPENWIRE keyword is used (instead of OPENWIRE) to enable this handling of general interconnect polygons, and OPENVIA is used to handle the wire regions containing VIAs. An example for layer derivations would be:

```
m1_open1 = DFM CRITICAL AREA m1 OPENWIRE S 0.5 OUTPUT S 1.0 S 2.0 D 0.25
m1_open2 = DFM CRITICAL AREA m1 OPENWIRE S 0.5 S 1.0 OUTPUT S 2.0 D 0.25
m1_open3 = DFM CRITICAL AREA m1 OPENWIRE S 0.5 S 1.0 S 2.0 OUTPUT D 0.25
```

```

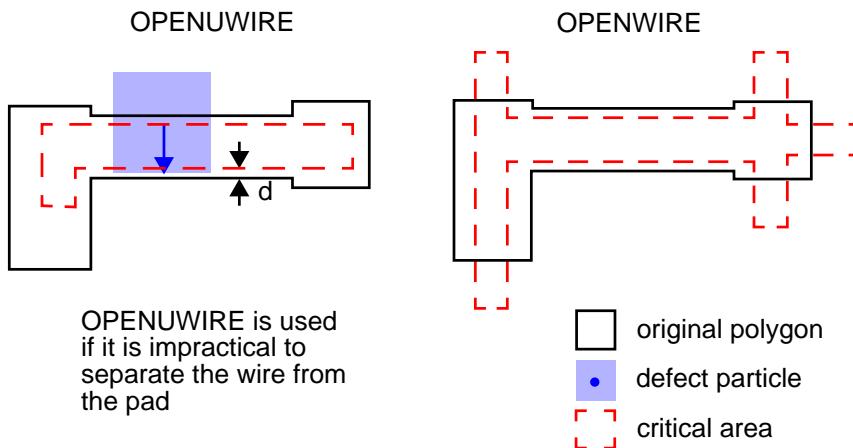
vial_open1 = DFM CRITICAL AREA vial OPENVIA bank_vial
            S 0.5 OUTPUT S 1.0 S 2.0 D 0.25
vial_open2 = DFM CRITICAL AREA vial OPENVIA bank_vial
            S 0.5 S 1.0 OUTPUT S 2.0 D 0.25
vial_open3 = DFM CRITICAL AREA vial OPENVIA bank_vial
            S 0.5 S 1.0 S 2.0 OUTPUT D 0.25
m1_crit_areal = m1_open1 OR vail_open1
m1_crit_area2 = m1_open2 OR vial_open2
m1_crit_area3 = m1_open3 OR vial_open3

```

The critical area region generated by the OPENWIRE keyword will generally be larger than for OPENUWIRE.

Comparing the two methods:

**Figure 4-51. OPENUWIRE Versus OPENWIRE**



#### Example 4

This example visually outlines the differences between OPENWIRE, OPENUWIRE, OPENVIA and the combination of OPENUWIRE and OPENVIA.

Assume:

- a square defect model with a radius of 0.2.
- a depth value (D) of zero. Note that this is for conceptual simplicity only. In practice, the depth is normally set to a value other than zero.
- you are interested in the critical area generated from potential opens on a metal layer (M1) and a via layer (VIA1).

Using these assumptions, you may write rule checks as follows:

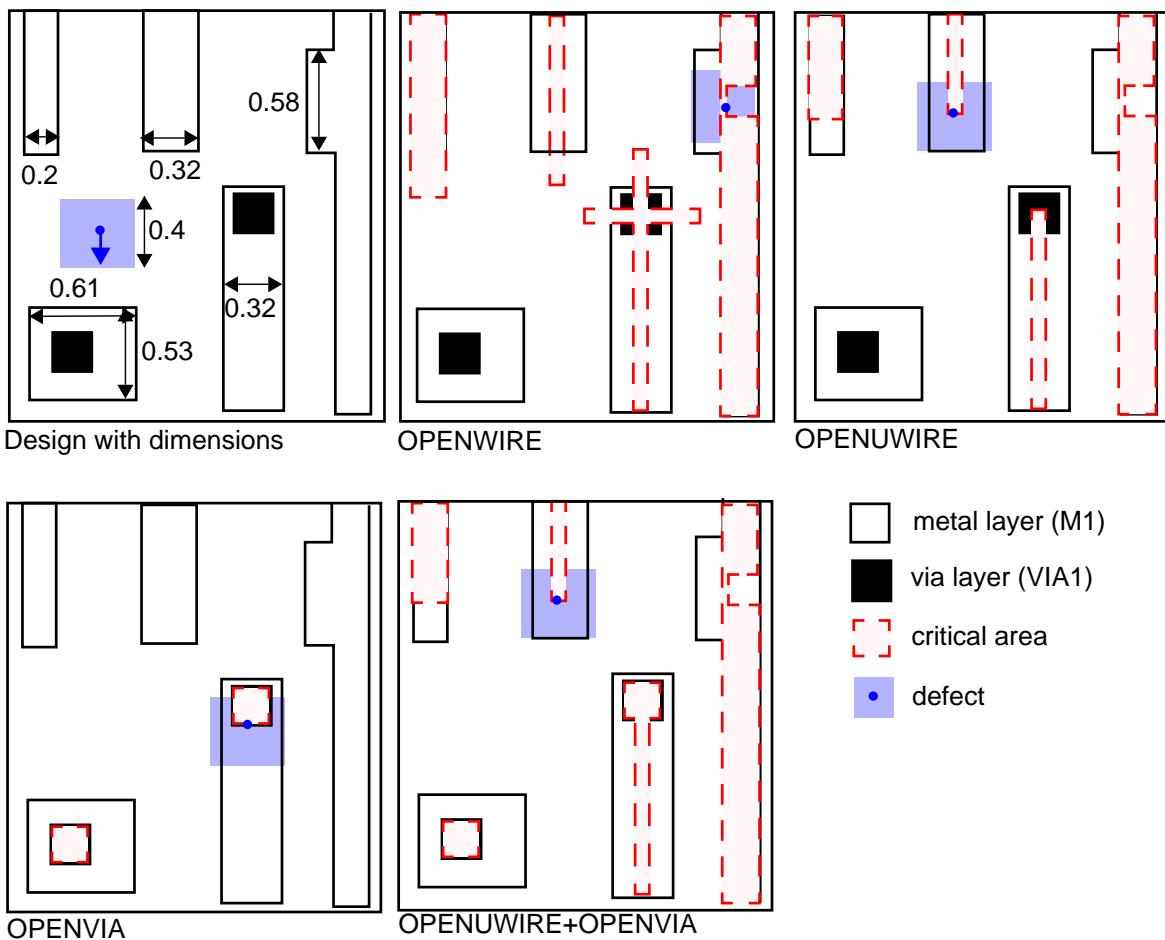
```

M1_OW_040 = DFM CRITICAL AREA M1 OPENWIRE S 0.20 D 0
M1_OU_040 = DFM CRITICAL AREA M1 OPENUWIRE S 0.20 D 0
M1_OV_040 = DFM CRITICAL AREA VIA1 OPENVIA S 0.20 D 0
M1_OUOV_040 = OR M1_OU_040 M1_OV_040

```

## DFM Critical Area

The following figure shows the critical area that is generated from these checks for a sample design.

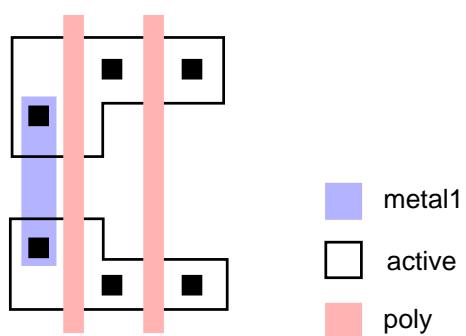


Note that for the OPENWIRE case, critical area is generated whenever the defect particle covers a line-end or causes an open on M1. For the OPENUWIRE case, by contrast, critical area is generated only if the defect causes an open on M1.

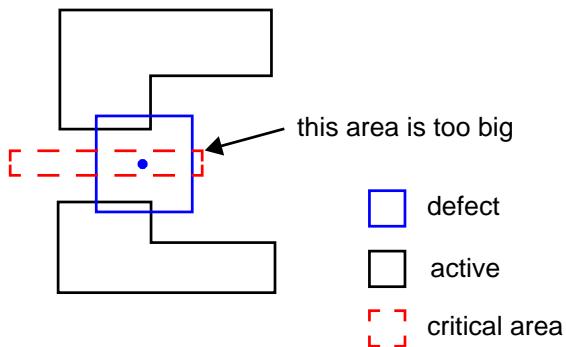
### Example 5

Excluding Gate Edges

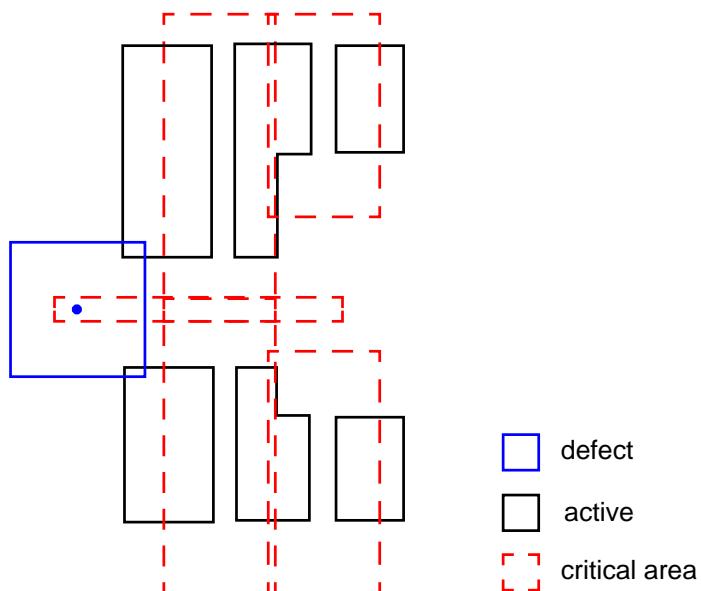
Assume you are checking for shorts on the active layer, specifically between gates.



If you perform a critical area analysis on the entire active layer, the results reports an area that is too large:



However, if you perform a critical area analysis on the source-drain regions ( $sd = \text{active NOT poly}$ ), the operation returns an even larger area that includes shorts between “false” edges on the  $sd$  layer:

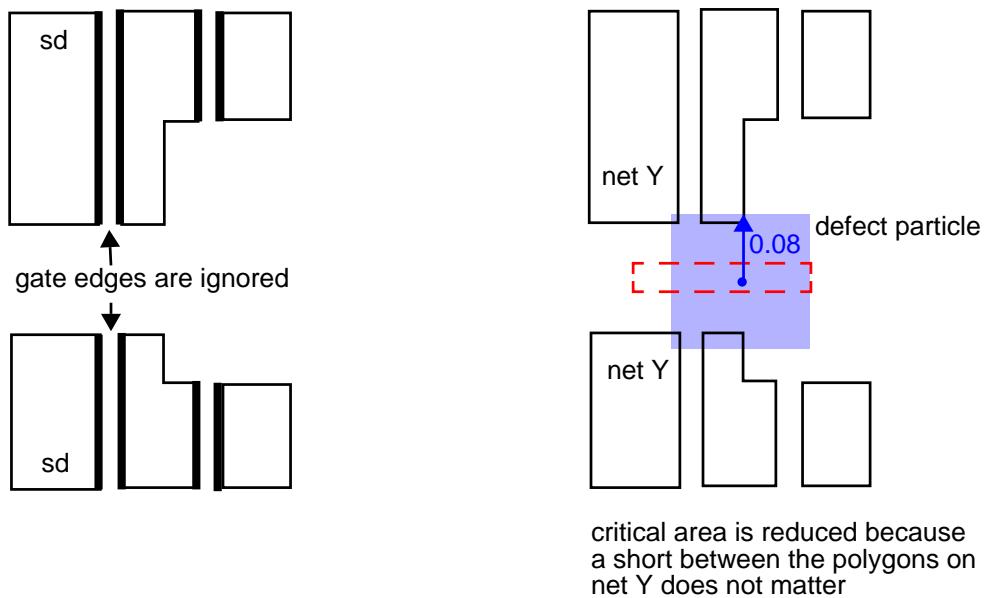


The correct way to calculate critical area in the gate region of the active layer is to check the  $sd$  layer, excluding the false edges. You do this by creating a gate layer that identifies false edges, then using the EXCLUDE EDGE keyword to exclude all edges coincident with the gate layer.

```
gate = active AND poly
sd = active NOT poly
CONNECT sd M1 BY contact
```

## DFM Critical Area

DFM CRITICAL AREA SHORT sd EXCLUDE EDGE gate S 0.080 OUTPUT



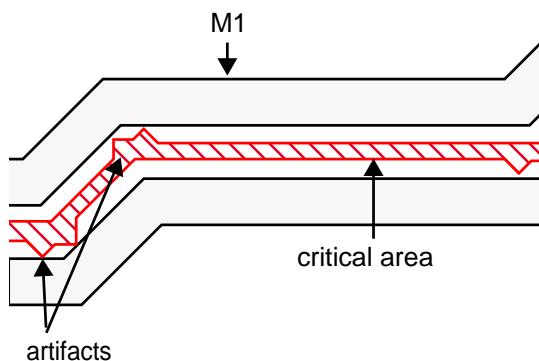
### Example 6

The example outlines how to use octagonal shapes to eliminate artifacts in your critical area regions.

In designs containing edges that are not vertical or horizontal, the square defect approximation can generate triangular artifacts in the critical area region. For example, the following rule uses a square with edge length 0.110 to model a defect particle in a design containing 45 degree edges:

SHORT\_M1\_SQ = DFM CRITICAL AREA M1 SHORT S 0.110

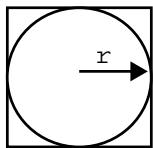
which produces the following critical area:



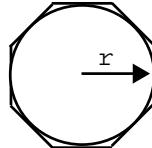
You can eliminate triangular artifacts for 45 degree edges by specifying the REGION OCTAGONAL keyword with the DFM Critical Area operation. This keyword uses equilateral octagons instead of squares to approximate defects when calculating critical area:

$$\text{side length} = 2r$$

$$\text{side length} = 2r\tan(\pi/8)$$



square defect approximation

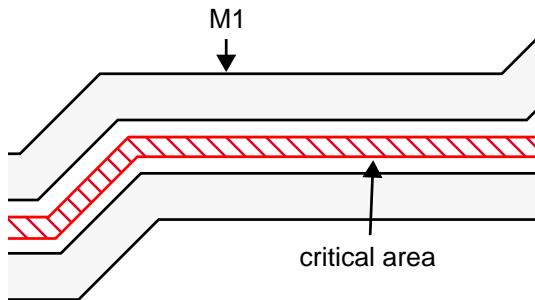


octagonal defect approximation

In the above example, the rule is modified as follows:

```
SHORT_M1_OCT = DFM CRITICAL AREA M1 SHORT REGION OCTAGONAL S 0.110
```

which produces following critical area:



Note that the elimination of artifacts is achieved at the expense of an increase in runtime. Also, note that using the octagonal approximation will only eliminate artifacts for 45 degree edges. Edges that are not horizontal, vertical, or at 45 degree angles will still produce artifacts.

# DFM Database

Specification statement

Syntax 1:

```
DFM DATABASE directory [OVERWRITE [REVISIONS]] '['{[DEVICES] [PINLOC]  
[ORIGINAL] [ANALYZE] [MEASURE] [PROPERTY] [DENSITY]  
[CONNECT]} | ALL [NOCONNECT]']'  
[ANNOTATE '[' "name" = "value" ']']...
```

Syntax 2:

```
DFM DATABASE DIRECTORY directory [OVERWRITE [REVISIONS]] '['{[DEVICES]  
[PINLOC] [ORIGINAL] [ANALYZE] [MEASURE] [PROPERTY] [DENSITY]  
[CONNECT]} | ALL [NOCONNECT]']'  
[ANNOTATE '[' "name" = "value" ']']...
```

## Summary

Syntax 1 creates a DFM database in the specified location. Syntax 2 creates a DFM database inside a specified parent directory, which allows you to control organization of one or more DFM databases from a file structure standpoint. Refer to the *Calibre YieldServer Reference Manual* for more information on DFM databases.

## Parameters

- ***directory***

If Syntax 1 is used, ***directory*** is a required argument that specifies the path to the directory created to contain the files that comprise the DFM database.

If Syntax 2 is used, ***directory*** is a required argument that specifies the path to the parent directory of the DFM database. The DFM database is created as a subdirectory within the parent directory, and is automatically named “\_%t\_.dfmdb”, where %t is the name of the top cell. If Layout Primary “\*” is specified in the rule file, the name of the actual top cell is used. If Layout Primary “name” is specified, the primary cell *name* is used. If the ***directory*** does not exist, it is created. If the ***directory*** already exists, it is not modified except to create or overwrite the DFM database subdirectory.

- **OVERWRITE**

An optional argument instructing the application to overwrite the DFM database directory if it already exists. If the given directory already exists, and the OVERWRITE argument is not provided, an error is issued. If the directory exists and the OVERWRITE argument is provided, the existing directory is removed and replaced with the new database directory. This option does not work if the DFM database contains any revisions other than the master revision.

- **REVISIONS**

An optional argument used with OVERWRITE that instructs the tool to overwrite the DFM database directory, even if it contains revisions.

- [ ]  
Syntactical elements required when additional parameters are supplied to define exactly which data gets written to the DFM database.  
Square brackets that are true syntactical elements appear in the syntax description as '[' and ']'.  
• DEVICES  
An optional keyword instructing the application to write device layers to the DFM database. Including this argument causes device extraction to be performed.  
• PINLOC  
An optional keyword that instructs the application to write pin location information to the DFM database.  
• ORIGINAL  
An optional keyword instructing the application to write original (merged) layers to the DFM database. Only those original layers that are used in a selected operation are saved. (That is, DFM Database ORIGINAL does not cause layers to be loaded that would not otherwise be loaded.)  
• ANALYZE  
An optional keyword instructing the application to save the input layers to all DFM Analyze operations.  
• MEASURE  
An optional keyword instructing the application to save the input layers to all DFM Measure operations.  
• PROPERTY  
An optional keyword instructing the application to save the input layers to all DFM Property operations.  
• DENSITY  
An optional keyword instructing the application to save the input layers to all Density operations.  
• CONNECT  
An optional keyword instructing the application to save all input layers involved in [Connect](#) statements to the DFM database. This is the default behavior when the rule file does not contain a [DRC Incremental Connect](#) YES statement. If DRC Incremental Connect YES statement is specified in the rule file, specifying CONNECT can result in a large number of database revisions. Note that CONNECT is also specified implicitly when ALL is specified.

- ALL

An optional keyword that is the same as specifying that all data is to be written to the DFM database. This is equivalent to specifying DEVICES, ORIGINAL, ANALYZE, MEASURE, PROPERTY, DENSITY, and CONNECT.

- NOCONNECT

An optional keyword instructing the application to not save all input layers involved in [Connect](#) statements to the DFM database. This is the default behavior when the rule file contains a [DRC Incremental Connect](#) YES statement. Connect layers can still be saved by other means, such as with [DFM RDB](#) or [Copy](#) statements, or as input layers to operations that appear as layer selectors in the DFM Database statement. NOCONNECT can be used with the ALL keyword.

- ANNOTATE '[' "name" = "value" ']

An optional keyword and arguments used to associate meta data with the entire database. Within the arguments for this keyword, the following constraints apply:

- The square brackets [ ] are required.
- The name can be a quoted or unquoted string, and need not be unique.
- The value must be a quoted string.

You can specify this keyword as many times as needed. Each occurrence adds another annotation (*name + value*) to the database.

## Description

Specifies the location and contents of the DFM database. The output layers of selected checks are always saved to the DFM database. Connect layers are always saved if connectivity extraction is performed.

Maximum result counts and maximum vertex counts are ignored when saving layers to the DFM database.

Refer to the [Calibre YieldServer Reference Manual](#) for more information on DFM databases.

## Examples

### Example 1

```
DFM DATABASE "/tmp/dfmdb" OVERWRITE REVISIONS
// Specifies the path to the DFM database as /tmp/dfmdb,
// and overwrites the database if it already exists, even if
// it contains revisions other than the master revision.
```

### Example 2

```
DFM DATABASE "full.dfmdb" OVERWRITE
ANNOTATE [ Purpose = "Complete run with all layers" ]
ANNOTATE [ Phase = "1" ]
// Specifies the path to the DFM database as ./full.dfmdb,
// and annotates it with two name/value pairs.
```

**Example 3**

```
DFM DATABASE DIRECTORY "/tmp/sample_design" OVERWRITE [ALL]
// Specifies the path to the DFM database parent directory
// as "/tmp/sample_design". The DFM database is created as a
// subdirectory within the parent directory with the naming
// convention "_%t_.dfmdb", where %t is the name of the top cell.
```

# DFM Defaults

Specification statement

## DFM DEFAULTS *operation options*

### Parameters

- *operation*

Required keyword that specifies the SVRF statement to which default *options* are applied. Currently only the DFM RDB statement is supported. The value of *operation* may be:

- RDB [GDS | OASIS] — applies default options to the DFM RDB statement. The GDS and OASIS keywords are used with DFM RDB GDS and DFM RDB OASIS.

- *options*

Required keyword set that specifies default options for the specified *operation* (which currently must be “RDB”). Values for this keyword set are:

- [FILE {filename | NOFILE}]  
[NODAL [ALL | OTHER | NULL]]  
[NOPSEUDO [NULL]] [NOEMPTY [NULL]]  
[OUTPUT [NULL]] [ALL CELLS [NULL]]  
[JOINED [NULL]]  
[RESULT CELLS {cell\_list | NULL}]  
[CHECKNAME {check\_name | NULL}]

which correspond to parameters in the DFM RDB statement. The NULL keyword cancels the associated default so that it no longer applies to subsequent instances of the *operation* (for the FILE option, the cancellation keyword is NOFILE, since NULL is a valid *filename* value in Calibre DFM applications).

### Description

Allows you to specify certain default *options* for the SVRF statement associated with the *operation*. Currently only the DFM RDB statement is supported.

The DFM Defaults statement applies only to the portion of the rule file after the statement, that is, to all subsequent instances of the specified *operation*. The DFM Defaults statement may be specified more than once — in this case, each subsequent statement can modify the defaults specified by earlier statements.

Any option specified in the operation itself takes precedence over the same option specified with DFM Defaults. If a default option is incompatible with a different, explicitly specified option, the default option is ignored. Therefore, a DFM Defaults statement cannot cause a subsequent *operation* to not compile.

### Examples

#### Example 1

Consider the following code:

```

CHECK1 { DFM RDB met1 "rdb1" }
DFM DEFAULTS RDB NODAL
CHECK2 { DFM RDB met2 "rdb2" }
CHECK3 { DFM RDB met3 "rdb3" NODAL ALL }
DFM DEFAULTS RDB NODAL NULL CHECKNAME "%_1_"
CHECK4 { DFM RDB met4 "rdb4" }
DFM DEFAULTS RDB ALL CELLS
CHECK5 { DFM RDB met5 "rdb5" }
CHECK6 { DFM RDB met6 "rdb6" NOPSEUDO }

```

The first DFM RDB operation has no options, either explicit or default. The second DFM RDB operation has the default option NODAL. The third DFM RDB operation has the explicit option NODAL ALL which is not compatible with the default NODAL; the latter is ignored. The second DFM Defaults statement cancels the NODAL default, so it no longer applies to the rest of the rule file. The fourth DFM RDB operation has the default option CHECKNAME "%\_1\_". The third DFM Defaults statement adds the ALL CELLS option to the previous set of defaults, and the fifth DFM RDB operation has both ALL CELLS and CHECKNAME options. The sixth DFM RDB operation has the same defaults and the explicit NOPSEUDO option. Since the NOPSEUDO option does not conflict with either ALL CELLS or CHECKNAME options, both default options apply.

The above code results in the following options for each of the DFM RDB operations:

```

CHECK1 { DFM RDB met1 "rdb1" }
CHECK2 { DFM RDB met2 "rdb2" NODAL }
CHECK3 { DFM RDB met3 "rdb3" NODAL ALL }
CHECK4 { DFM RDB met4 "rdb4" CHECKNAME "%_1_" }
CHECK5 { DFM RDB met5 "rdb5" ALL CELLS CHECKNAME "%_1_" }
CHECK6 { DFM RDB met6 "rdb6" ALL CELLS NOPSEUDO CHECKNAME "%_1_" }

```

## Example 2

Consider the following code:

```

CHECK1 { DFM RDB met1 "rdb1" }
DFM DEFAULTS RDB FILE "rdb2"
CHECK2 { DFM RDB met2 }
CHECK3 { DFM RDB met3 "rdb3" }
DFM DEFAULTS RDB NOFILE
CHECK4 { DFM RDB met4 }
CHECK5 { DFM RDB met5 "rdb5" }
DFM DEFAULTS RDB FILE NULL
CHECK6 { DFM RDB met6 }

```

The first DFM RDB operation has an explicit file name; no defaults have been specified. The second DFM RDB operation uses the default file name "rdb2". The third DFM RDB operation has an explicit file name "rdb3"; the default "rdb2" is still in effect but is not used since an explicit file name is specified. The second DFM Defaults statement cancels the FILE default, so that it no longer applies to the rest of the rule file. The fourth DFM RDB operation is invalid because it has no file name and no default file name is defined; this operation will not compile. The fifth DFM RDB operation has an explicit file name. The sixth DFM RDB operation uses the default file name NULL; this operation does not create an ASCII RDB file, but can be used in Calibre DFM to save a layer to the DFM database.

## DFM Defaults

---

The above rules file results in the following options for each of the DFM RDB operations (note that the DFM RDB operation in CHECK4 is not valid):

```
CHECK1 { DFM RDB met1 "rdb1" }
CHECK2 { DFM RDB met2 "rdb2" }
CHECK3 { DFM RDB met3 "rdb3" }
CHECK4 { DFM RDB met4 }
CHECK5 { DFM RDB met5 "rdb5" }
CHECK6 { DFM RDB met6 NULL }
```

### Example 3

This example shows how DFM Defaults can be used with DFM RDB GDS and DFM RDB OASIS. In the following code, layers met1 and met2 are written to out.gds.

```
DFM DEFAULTS RDB GDS FILE out.gds
CHECK1 { DFM RDB GDS DEFAULT met1 met2 }
```

## DFM Expand Edge

Layer operation

```
DFM EXPAND EDGE layer1
{OUTSIDE BY outside_property |
INSIDE BY inside_property |
OUTSIDE BY outside_property INSIDE BY inside_property}
[EXTEND BY extend_property]
[OFFSET offset_property]
[UNMERGED]
```

Used only in Calibre YieldEnhancer applications.

### Parameters

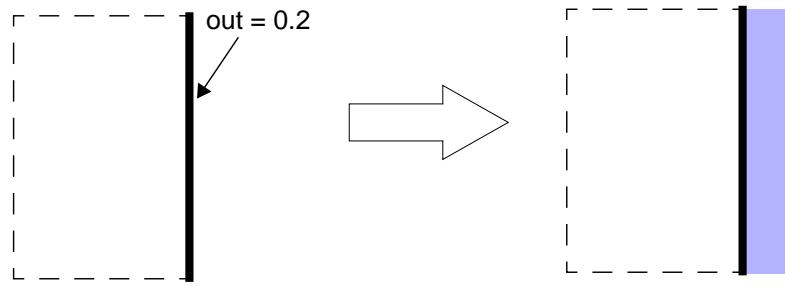
- ***layer1***

A required input edge or edge cluster layer. If this layer is created using a [DFM Property](#) operation, the names of properties attached to this layer can be passed to the operation as the arguments *outside\_property*, *inside\_property*, *extend\_property*, and *offset\_property*. Alternatively, any of these arguments may be specified as constant values (either directly in the operation or by using a Variable statement). If *layer1* is an edge cluster, each edge in the cluster is expanded by the same amount.

If *layer1* is an unmerged edge or error layer that does not contain DFM properties, it is merged before expansion occurs. If an unmerged *layer1* does contain DFM properties, it is not merged before expansion occurs. The UNMERGED keyword controls merging of the output layer.

- **OUTSIDE BY *outside\_property***

A required keyword and argument that specifies that each edge is expanded in the direction perpendicular to the edge and outside of the original polygon from which it was created. The name of the property that defines the expansion distance is specified by *outside\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *outside\_property*.

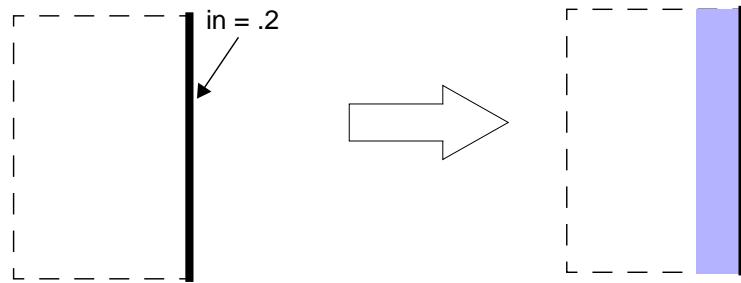


`new_layer = DFM EXPAND EDGE input_layer  
OUTSIDE BY "out"`

  original polygon  
  input\_layer  
  new\_layer

- **INSIDE BY *inside\_property***

A required keyword that specifies that each edge is expanded in the direction perpendicular to the edge and inside of the original polygon from which it was created. The name of the property that defines the expansion distance is specified by *inside\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *inside\_property*.

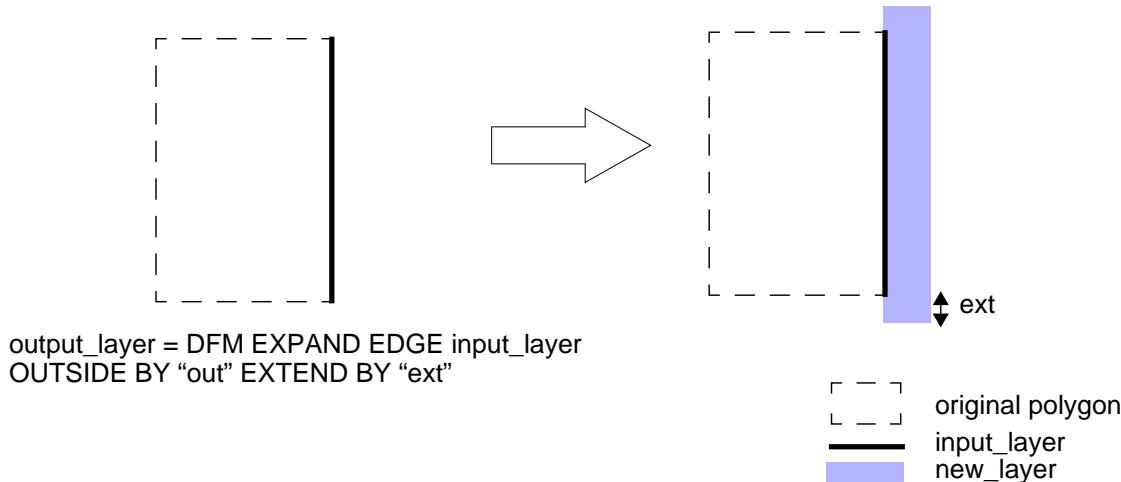


`new_layer = DFM EXPAND EDGE input_layer  
INSIDE BY "in"`

  original polygon  
  input\_layer  
  new\_layer

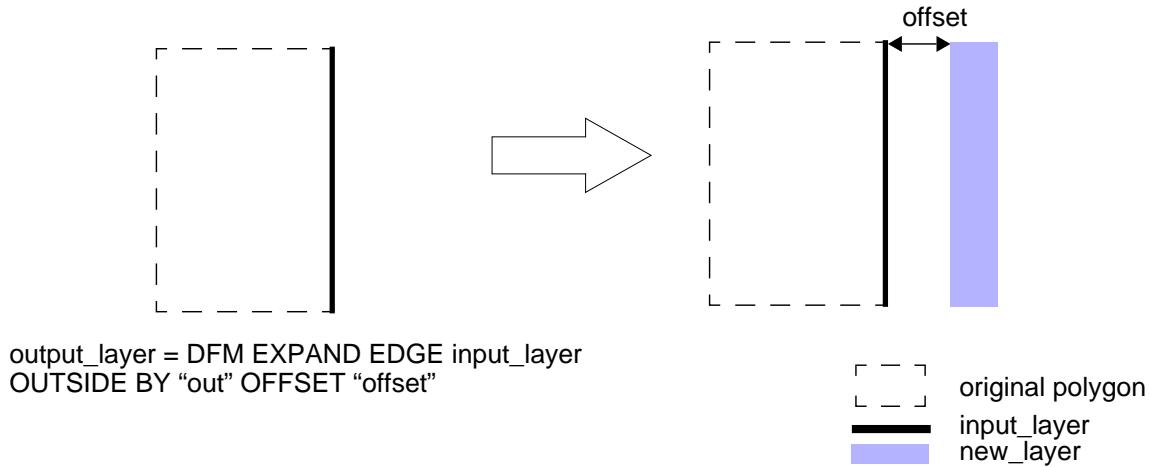
- **EXTEND BY *extend\_property***

An optional keyword that specifies that each edge is extended by *extend\_property* in both directions along the edge before expansions are applied. The name of the property that defines the expansion distance is specified by *extend\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *extend\_property*.



- **OFFSET *offset\_property***

Optional keyword that causes the expanded polygon to be shifted from the original edge. The name of the property that defines the offset value is specified by *offset\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *offset\_property*. Edge expansion with an offset is illustrated in the following figure:



If the original edge is expanded in one direction, for example OUTSIDE, the edge is first expanded by the OUTSIDE value then shifted by the OFFSET value in the direction of the expansion, creating the gap shown in the figure.

Note that the edge can be expanded only in one direction if OFFSET is used: either INSIDE BY or OUTSIDE BY must be specified, but not both at the same time.

- UNMERGED

An optional keyword that causes the operation to produce an output layer that contains unmerged polygons (that is, individual polygons created from each edge are preserved, even if they overlap). The output layer is annotated with DFM properties copied from the input layer (if they exist). Every output polygon contains the property values from the input edge used to construct the polygon.

Unmerged polygon layers can be processed by the [DFM Copy](#), [DFM Property Merge](#) and [DFM RDB](#) operations.

### Description

Expands edges into rectangles. This operation is similar to [Expand Edge](#). The primary differences between this operation and Expand Edge are as follows:

- DFM Expand Edge can take expansion values from DFM properties attached to *layer1* instead of from values specified with keywords in the operation.
- DFM Expand Edge does not support the secondary keywords FACTOR or CORNER FILL.
- DFM Expand Edge can accept derived error layers as input.

At least one of the options **OUTSIDE BY** and **INSIDE BY** must be specified; if either of these is not specified the corresponding expansion is not applied (its value is zero). The values of **OUTSIDE BY**, **INSIDE BY** and **OFFSET** must be positive.

The constant values or the values of the property that define the extension distances, *extend\_property*, can be positive or negative; a negative value causes an edge to shrink (pull in from both ends) instead of extend in both directions along the edge.

The result of this operation is a derived polygon layer. As with all derived polygon layers, overlapping or touching polygons are merged, so it is possible that polygons created for unrelated edges will be merged together.

This operation does not require or preserve connectivity. The resulting derived polygon layer does not contain net numbers, even if *layer1* contains connectivity.

### Examples

#### Example 1

This example shows how to use a layer created with DFM Property in a DFM Expand Edge operation.

```
met_edge = m1 LENGTH < 1
met_edge_p = DFM PROPERTY met_edge [ OUT = LENGTH(met_edge)*.01 ]
[ IN = LENGTH(met_edge)*.02 ]
met_expand = DFM EXPAND EDGE met_edge_p OUTSIDE BY "OUT" INSIDE BY "IN"
```

**Example 2**

In this example, an edge layer e is annotated with several DFM properties. One of these properties is used to expand the edge layer while preserving the properties. The unmerged layer is then processed by the DFM Property Merge operation, which computes properties for the merged polygons from their unmerged component polygons according to the specified expression. Specifically, the property A on the output layer exm contains the area of the largest polygon on layer p overlapping the edge from which the output polygon was created.

```
e_p = DFM PROPERTY e p OVERLAP [ S = AREA(p)/LENGTH(e) ] [ A = AREA(p) ]
ex = DFM EXPAND EDGE e_p BY "S" UNMERGED
exm = DFM PROPERTY MERGE ex [ A = MAX( PROPERTY(ex, "A") ) ]
```

# DFM Expand Enclosure

Layer operation

**DFM EXPAND ENCLOSURE** *via\_layer\_in conn\_layer\_in*

**SIDE** *side\_expansion*  
**LINE** *line\_expansion*  
**END** *end\_expansion*  
{**LINEENDMAX** *line\_enclosure\_dist* | **SIDEMAX** *side\_enclosure\_distance*}  
**SPACE** *min\_space* **STEP** *step\_size*  
[**SPACING** *space space\_layer*] ...

Used only in Calibre YieldEnhancer applications.

## Summary

Layer operation used to generate regions of expanded enclosure for vias. Specifically, it creates a new derived polygon layer containing the regions representing these expanded enclosures. The operation preserves node numbers, so node numbers on the via layer are maintained on the output layer.

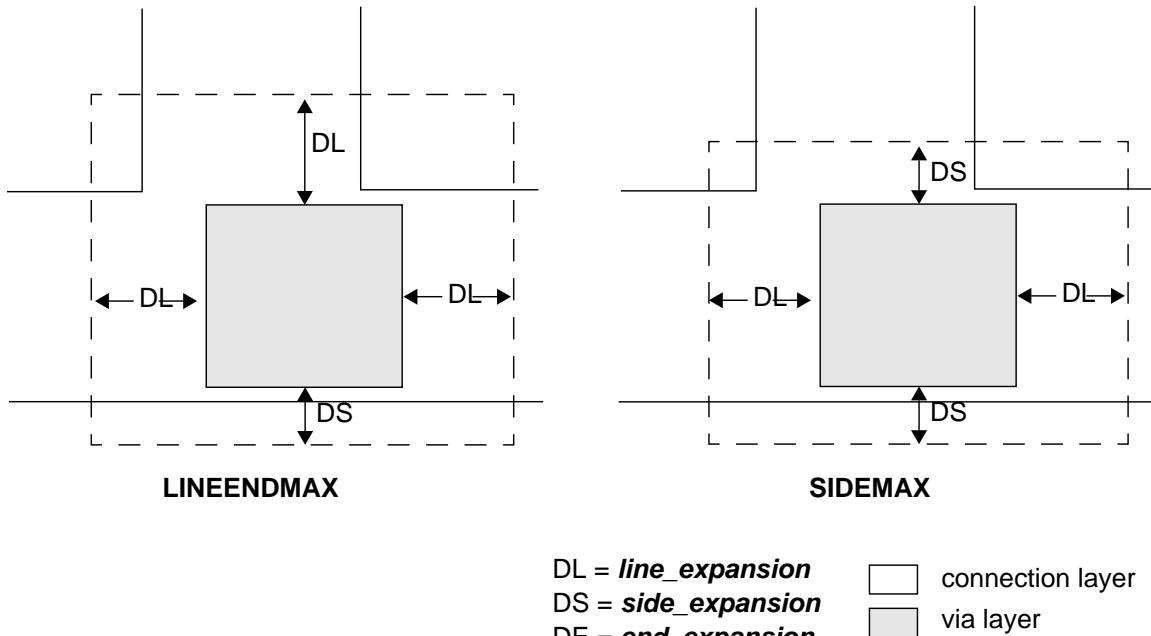
## Parameters

- ***via\_layer\_in***  
A required original or derived polygon layer. This is a via layer.
- ***conn\_layer\_in***  
A required original or derived polygon layer. This is a connection layer.
- **SIDE** *side\_expansion*  
A required keyword and value specifying the expansion distance, in microns, to be applied to edges classified as SIDEs.
- **LINE** *line\_expansion*  
A required keyword and value specifying the expansion distance, in microns, to be applied to edges classified as LINEs.
- **END** *end\_expansion*  
A required keyword and value specifying the expansion distance, in microns, to be applied to edges classified as ENDs.
- **LINEENDMAX** *line\_enclosure\_dist* | **SIDEMAX** *side\_enclosure\_distance*  
A required keyword and value set that defines the distance in microns used to classify via edges as being LINE, END, or SIDE.
  - An edge is a LINE if the enclosure  $\geq$  *line\_enclosure\_dist*
  - An edge is a SIDE or END if enclosure  $<$  *line\_enclosure\_dist*

The behavior of **SIDEMAX** is identical to that of **LINEENDMAX** except in cases where vias are enclosed by different values. In these cases, the minimum enclosure distance is used

as the *side\_enclosure\_distance* value (LINEENDMAX uses the maximum enclosure distance for the *line\_enclosure\_dist* value). Refer to [Figure 4-52](#).

**Figure 4-52. LINEENDMAX Versus SIDEMAX**



- **SPACE *min\_space***

A required keyword and value specifying the minimum spacing allowed between an enclosure and an edge on the connection layer.

- **STEP *step\_size***

A required keyword and floating-point number that specify the enclosure increment, in user units. This is the smallest distance by which an enclosure geometry can extend beyond the via it encloses. In situations where the an expansion cannot be satisfied, the operation attempts to find the greatest expansion that meets the design rules and is an integer multiple of the STEP value.

- **SPACING *space space\_layer...***

An optional keyword and associated values defining additional spacing constraints placed on the enclosure regions. The first value, *space*, defines the minimum allowed spacing, in microns. The second value, *space\_layer*, defines the layer against which the spacing is measured. The *space\_layer* can be any polygon or edge layer. More than one of these constructs is allowed.

### Description

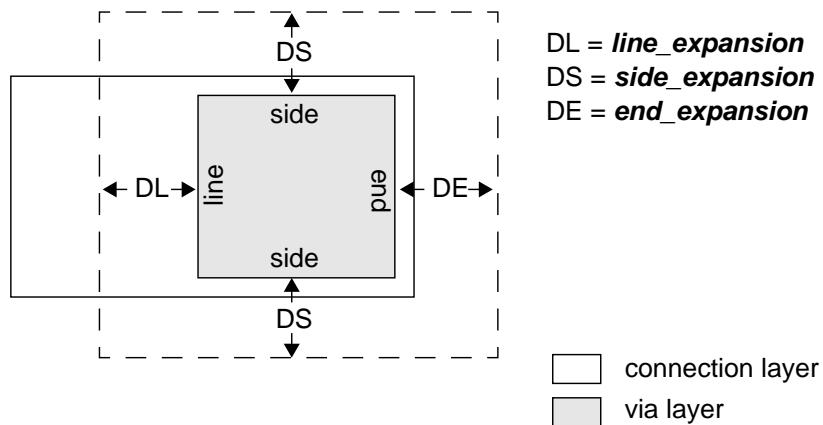
Generates expanded enclosure regions for orthogonal via edges. This operation does not operate on 45 degree or skew edges. It classifies each of the via edges as being a SIDE, LINE, or END, then expands those edges by the expansion distance specified for that class of edge. If the full

expansion would result in a spacing violation, it expands by a smaller distance that does not create a violation.

### Edge Classification

Edge classification is based on two factors:

- The size of the original enclosure, that is, the distance by which the connection geometry extends beyond the via. The operation compares the enclosure to the **LINEENDMAX** value. If the enclosure is  $\geq \text{LINEENDMAX}$ , the edge is a LINE. Otherwise the edge is a SIDE or an END.
- The classification of the adjacent and opposite edge(s). If an edge is  $< \text{LINEENDMAX}$ , and both adjacent edges are SIDES and the opposite edge is a LINE, then the edge is an END. In all other cases where an edge is  $< \text{LINEENDMAX}$ , the edge is a SIDE.



### Expansion

Once the edges have been classified, the operation generates the enclosure region by extending each via edge by the enclosure distance specified for the edge type:

- **SIDE** — *side\_dist*
- **LINE** — *line\_dist*
- **END** — *end\_dist*

Note that the operation only generates expanded enclosures for orthogonal edges. It does not operate on 45 degree or skew edges.

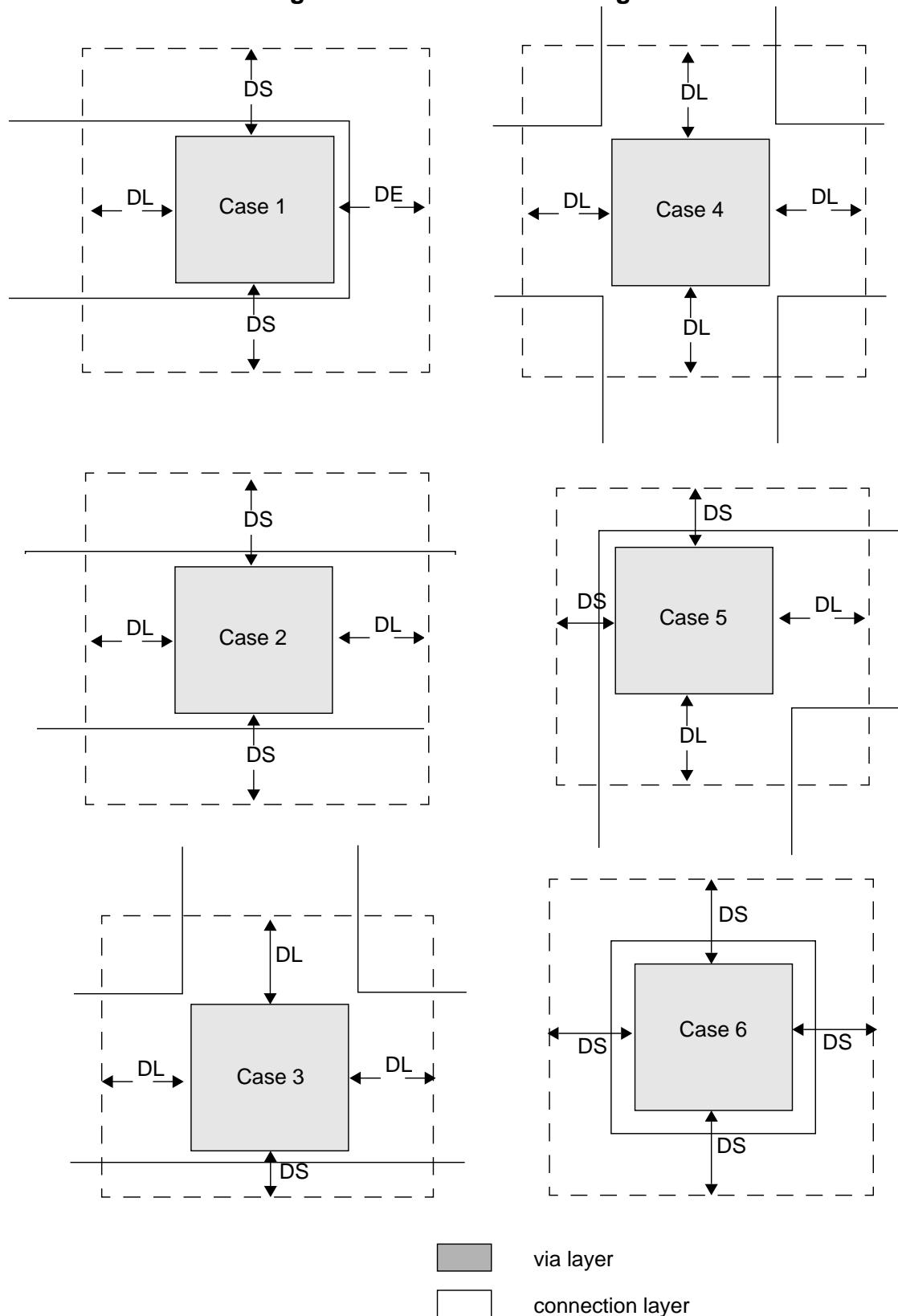
Expansions must conform to the spacing rules defined for the operation. At least one spacing rule applies. This is the rule you define using the SPACE keyword, and it defines the minimum spacing between an expanding enclosure and features on the connection layer.

You can define as many additional spacing rules as needed using the optional SPACING keyword. Each one defines the minimum spacing between an enclosure and the features on the specified polygon or edge layer. Typically, spacing layers are subsets of the connection layer, for example, orthogonal metal edges may have one spacing while non-orthogonal metal edges may have a different (larger) spacing.

In situations where the an expansion cannot be satisfied without violating a spacing rule, the operation expands by a smaller distance that does not create a violation. The operation attempts to find the greatest expansion that meets the design rules and is an integer multiple of the STEP value.

### DFM Expand Enclosure Geometric Algorithms

DFM Expand Enclosure covers six specific orthogonal configurations, shown in [Figure 4-53](#) with SIDE, LINE and END displacement marked as DS, DL and DE, respectively.

**Figure 4-53. Enclosure Configurations**

## DFM Fill

Layer operation

**DFM FILL** *spec\_name* [*output\_name* ...] [COLOR *color\_name*] {[HLAYER] COMPRESS | COMPRESS [*cell\_name\_prefix*] | ENVELOPE *value*}

Used only in Calibre YieldEnhancer applications.

### Parameters

- *spec\_name*

A required argument indicating the fill specification to use when generating the fill layer.

- *output\_name*

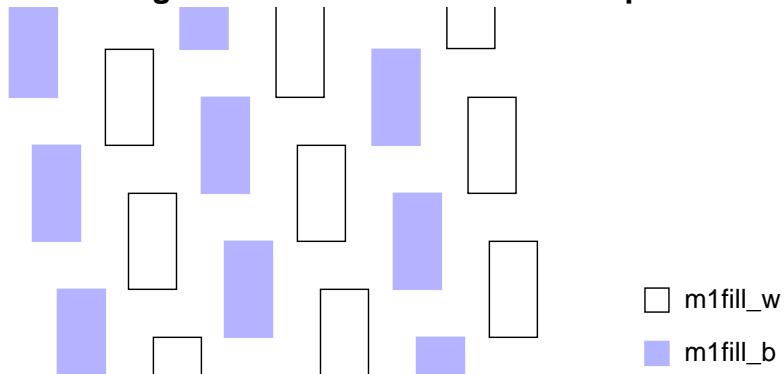
An optional argument specifying the fill geometries to be selected for output by this operation. These outputs must be defined in the DFM Spec Fill statement for *spec\_name*. If multiple *output\_names* are specified, the data from all *output\_names* are merged into a single output layer for the operation.

If no *output\_name* is specified, all output generated based on *spec\_name* is merged into a single output layer, regardless of whether an *output\_name* was assigned to individual shapes in the **DFM Spec Fill** statement or not.

- COLOR *color\_name*

An optional keyword that enables splitting of fill output into two layers. The splitting is done such that shapes from the input layer are written to each output layer in an alternating pattern (refer to [Figure 4-54](#)). This is also known as double patterning.

**Figure 4-54. DFM Fill COLOR Output**



m1fill\_w = DFM FILL "m1fill\_spec" m1 COLOR "white"

m1fill\_b = DFM FILL "m1fill\_spec" m1 COLOR "blue"

For double patterning, you must specify two DFM Fill operations with the same *spec\_name* and *output\_name* values, and different *color\_name* and output layer values. The *color\_name* is an arbitrary user-defined string. The COLOR keyword is not valid with STRETCHFILL shapes or the FILLSTACK keyword.

- [HLAYER] COMPRESS | COMPRESS [*cell\_name\_prefix*]

An optional keyword set designed to reduce file sizes when the output layer is written using the DFM RDB operation. When COMPRESS is specified, the operation replaces repeating fill patterns (composed of up to 42 fill shapes) with an equivalent cell placement. The output layer is intended to be used only with the DFM RDB operation with GDS or OASIS specified.

The HLAYER keyword specifies that the output layer is the hierarchy layer for a set of multi-output or single-output DFM Fill operations. The hierarchy layer contains the hierarchy (placement) information for all layers in the set. Exactly one hierarchy layer must be specified for each set of multi-output or single-output DFM Fill operations. Additionally, the hierarchy layer must be included when writing one or more output layers using DFM RDB GDS or OASIS. The *output\_name* and *cell\_name\_prefix* arguments are not valid with the HLAYER keyword. The HLAYER keyword must both be specified with and appear before COMPRESS.

The *cell\_name\_prefix* argument allows you to specify a prefix for cell names in a particular fill shape pattern. This argument is useful for avoiding naming conflicts that can occur when generating cell names. Multiple instances of *cell\_name\_prefix* must be unique for each DFM Spec Fill statement, and identical within a single DFM Spec Fill statement. The default begins with “\_A”, and increments alphabetically to “\_Z”. If more than 26 DFM Spec Fill statements are present in a rule file, a number is appended to the default prefix. Cell names are generated using a combination of the cell name prefix and an alphanumeric hash string. The hash string represents the specific combination of fill shapes, the number of repetitions, and orientation information. If a *cell\_name\_prefix* is specified, the COMPRESS keyword must be specified at the end of the statement to prevent ambiguity.

The COMPRESS keyword set is valid only in hierarchical nmDRC mode (running in DFM mode produces a compilation error). In flat nmDRC mode, an uncompressed layer is produced that cannot be written using DFM RDB GDS or OASIS.

- ENVELOPE *value*

An optional keyword set that generates a fill pattern enclosure layer by expanding *spec\_name* shapes by the specified *value*. The following two operations are geometrically equivalent:

```
env_layer1 = SIZE (DFM FILL spec) BY 0.1  
env_layer2 = DFM FILL spec ENVELOPE 0.1
```

The ENVELOPE keyword, when used in conjunction with the COMPRESS keyword, can provide improved run-time and memory utilization by reducing the number of geometries on the output layer. The specified *value* must be greater than half the step distance between fill shapes. When OFFSET is used, the *value* must be large enough to cover increased corner to corner distances. For certain high-density designs, ENVELOPE may not provide significant improvements.

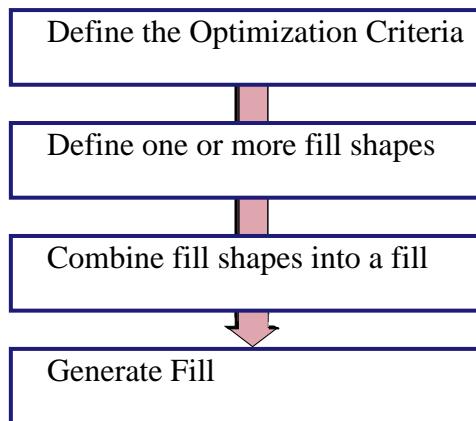
## Description

DFM Fill is an SVRF operation that creates an output layer consisting of fill polygons.

The output of this operation is a derived polygon layer containing the generated fill. When adding fill shapes, the DFM Fill operation treats the design as if it were flat and creates the initial fill shapes at the top level of the hierarchy. The operation then cleans up the fill shapes and, wherever applicable, pushes repeated shapes lower in the hierarchy, thereby reducing the final design file size and improving performance of future operations.

The sizes, shapes, and positions of these fill polygons are controlled through a fill specification, which you must define through the [DFM Spec Fill](#) statement. [Figure 4-55](#) shows the basic use model for this operation. For a complete discussion, refer to “[Adding Fill](#)” in the [Calibre YieldAnalyzer and YieldEnhancer Reference Manual](#).

**Figure 4-55. Typical Use Model for DFM Fill**



All DFM Fill operations referencing the same DFM Spec Fill are processed concurrently.

## Examples

This example shows how the COMPRESS keyword is used with a set of multi-output DFM Fill operations, and with a single-output operation.

In the following code, there are three output layers (m1\_fill, m2\_fill, m3\_fill) and a hierarchy layer (hier\_layer). All four layers are written to a GDS file using a DFM RDB GDS operation. Layer m1\_fill is written to layer 81, m2\_fill is written to layer 82, and so on.

```

m1_fill = DFM FILL M1_M2_M3_FILL M1 COMPRESS
m2_fill = DFM FILL M1_M2_M3_FILL M2 COMPRESS
m3_fill = DFM FILL M1_M2_M3_FILL M3 COMPRESS
hier_layer = DFM FILL M1_M2_M3_FILL HLAYER COMPRESS

write_gds {
    DFM RDB GDS "compressed_fill.gds" m1_fill 81 m2_fill 82 m3_fill 83
    hier_layer 0
}
  
```

Single output (with prefix specified):

```

m1fill_compress = DFM FILL X COMPRESS "_M1"
hier_layer = DFM FILL X HLAYER COMPRESS "_M1"

write_gds {
  
```

## DFM Fill

---

```
    DFM RDB GDS "compressed_fill.gds" m1fill_compress 81 hier_layer 0  
}
```

# DFM Function

Specification statement

## DFM FUNCTION

```
'['
function_name '(' type argument [',' type argument ...])'
{TABLE [LINEAR | SPLINE] '{' input1 result1 [input2 result2 ...] '}' |
expression |
MATRIX matrix_specification |
TVF_NUMBerg_FUNCTION('procedure_name ','library_name')' }
']'
```

### Summary

Creates a user-defined function that can be used in expressions for [DFM Analyze](#), [DFM Property](#) or other DFM Function definitions.

### Parameters

The entire parameter block for this statement must be enclosed in brackets [ ].

- *function\_name*

A required argument assigning a name to the function being defined. The name of the new function must not coincide with the name of any other DFM function or any built-in function allowed in DFM Analyze (such as AREA, EC, EXP, and so on).

- *(' type argument ')*

A required argument pair specifying the type(s) of data on which this function operates and assigning a name by which to reference the data. When the function is defined by a TABLE, only one *type argument* pair is allowed, and the type must be NUMBER. When the function is defined by an expression, you can supply an arbitrary number of arguments which can be of any type. Multiple *type argument* pairs must be separated by commas.

The type of the argument restricts valid uses of this argument in the body of the function. The *type* must be one of:

- NUMBER — a numeric value, used either as a number or as an argument to a function that accepts only numbers as arguments.
- LAYER — a string used as an argument to a function that accepts only layer names as arguments.
- STRING — a string used as the name of a parametrized property.
- VECTOR — a two-dimensional array of numbers in which one dimension, the tuple size, is fixed and the other dimension, the number of tuples, is variable.
- VSTRING — a two-dimensional array of strings in which one dimension, the tuple size, is fixed and the other dimension, the number of tuples, is variable.
- NETID — specifies that the associated argument is of type netID.

- **VNETID** — specifies that the associated argument is of type vnetID

The set of type/argument pairs must be enclosed in parentheses ( ).

- **TABLE [LINEAR | SPLINE] '{' *input1 result1* [*input2 result2* ...] '}'**

A keyword and associated list defining a set of ordered pairs forming a lookup table for calculating function results. When you specify TABLE, the following must be true:

- The list that defines the table must be enclosed in curly brackets { }.
- The function *inputs* (first values in the ordered pairs) in the table must be supplied in monotonically increasing order.

The secondary keywords, LINEAR and SPLINE control how functions values are calculated when the input value is between two adjacent input values:

- LINEAR — piecewise LINEAR interpolation algorithm. This is the default algorithm.
- SPLINE — bicubic interpolation algorithm.

When using a table lookup for calculating function results, the *type* must be NUMBER. The result is a floating-point number.

---

**Note**

For argument values outside of the range covered by the table, DFM Function will extrapolate the function based on the closest table points. This extrapolation is especially problematic for spline functions, but even for piece-wise linear functions it is unlikely to be accurate. You are encouraged to always protect interpolated functions from out-of-range arguments using [Conditional Expressions](#).

---

- ***expression***

An expression that performs a calculation involving layer data. You must supply either a TABLE, an *expression*, a MATRIX specification, or a TVF\_NUMber\_FUNction specification.

The *expression* may involve numbers (including numeric variables), [Unary Operators](#) (+, -, !, ~), [Binary Operators](#) (^, \*, /, +, -), parentheses ( ), [DFM Functions](#), [DFM Properties](#), [Conditional Expressions](#), and most [Measurement Functions](#). Note that SUM and PRODUCT are not supported in these expressions.

The usual rules of operator precedence apply, and parentheses may be used to establish precedence in the calculation.

- If an expression calls other DFM Function functions, the declared argument types match.
- Expressions for DFM Function cannot call other DFM Functions recursively, regardless of whether the recursion is finite or infinite.

Additional restrictions may come from the context in which the function is used, for example, a function containing AREA(L) will be invalid if called in DFM Analyze expression inside the SUM() function. For more information on expressions, refer to [DFM Expressions](#).

- **MATRIX *matrix\_specification***

Used for interpolation of multi-dimensional tables. The ***matrix\_specification*** contains the multi-dimensional table that defines the function. A function of N variables F(X1, X2, ... XN) is defined by a set of values which must be specified on a regular grid of coordinates X1, ... XN. The format of the ***matrix specification*** is as follows:

```
'{ ' X11 [,] X12 [,] ... X1p '}' [,]
'{ ' X21 [,] X22 [,] ... X2q '}' [,]
'{ ' X31 [,] X32 [,] ... X3r '}' [,]
.
.
.
'{ ' XN1 [,] XN2 [,] ... XNs '}' ::

'{ ' F(X11, X21, X31, ... XN1) [,]
    F(X12, X21, X31, ... XN1) [,]
.
.
.
    F(X1p, X21, X31, ... XN1) '}' [,]

'{ ' F(X11, X22, X31, ... XN1) [,]
    F(X12, X22, X31, ... XN1) [,]
.
.
.
    F(X1p, X22, X31, ... XN1) '}' [,]
.
.

'{ ' F(X11, X2q, X31, ... XN1) [,]
    F(X12, X2q, X31, ... XN1) [,]
.
.
.
    F(X1p, X2q, X31, ... XN1) '}' [,]

'{ ' F(X11, X21, X32, ... XN1) [,]
    F(X12, X21, X32, ... XN1) [,]
.
.
.
    F(X1p, X21, X32, ... XN1) '}' [,]

'{ ' F(X11, X22, X32, ... XN1) [,]
    F(X12, X22, X32, ... XN1) [,]
.
.
.
    F(X1p, X22, X32, ... XN1) '}' [,]
.
.

'{ ' F(X11, X2q, X32, ... XN1) [,]
    F(X12, X2q, X32, ... XN1) [,]
.
.
.
    F(X1p, X2q, X32, ... XN1) '}' [,]

'{ ' F(X11, X21, X31, ... XNs) [,]
    F(X12, X21, X31, ... XNs) [,]
.
.
.
    F(X1p, X21, X31, ... XNs) '}' [,]

'{ ' F(X11, X22, X31, ... XNs) [,]
```

```

F(X12, X22, X31, ... XNs) [,]
  . . .
F(X1p, X23, X31, ... XNs) '}' [,]
  . . .

'{ F(X11, X2q, X3r, ... XNs) [,]
F(X12, X2q, X3r, ... XNs) [,]
  . . .
F(X1p, X2q, X3r, ... XNs) '}'

```

In this specification, the braces enclosed in tick marks, '{' and '}', are literal and must be present in the rule file. These braces delimit the “rows” of the matrix (the line breaks are ignored, as is the case with SVRF in general). The commas in square brackets can be used to improve readability but are syntactically optional.

For an N-dimensional table, the first N rows contain the coordinates of the grid points on which the function is specified. The first row contains the values of the first coordinate, X1: X11, X12, ... X1p, where p is the number of table points in the first dimension. Similarly, the remaining rows from the second to the Nth contain the coordinates of the grid points in other dimensions. All coordinates must be specified in monotonically increasing order. That is,  $X11 < X12 < X13 < \dots < X1p$ .

The function values are separated from the coordinates by double colons “::”. The function values are specified in the “row order”: the first row of values contains the function values for all values of coordinate X1 while coordinates X2 ... XN are fixed at their lowest values. The second row again goes over all values of X1, X2 is advanced to its second value X22, and the rest of the coordinates are at their lowest values. After all function values for all values of X1 and X2 are specified this way, the procedure is repeated for X3 set to its second value X32, and so on. The last row contains function values for all values of X1 and the largest values of X2 ... XN.

For example, the following function specifies a 3-dimension table on a 3x3x3 grid:

```

DFM FUNCTION [ F( NUMBER X, NUMBER Y, NUMBER Z )
  MATRIX
    { 1, 2, 3 }
    { 4, 5, 6 }
    { 7, 8, 9 } :::
    { 147 247, 347 }
    { 157 257, 357 }
    { 167 267, 367 }
    { 148 248, 348 }
    { 158 258, 358 }
    { 168 268, 368 }
    { 149 249, 349 }
    { 159 259, 359 }
    { 169 269, 369 }
]

```

This table specifies the following function values:

```

F(x1, y1, z1) = F(1, 4, 7) = 147
F(x2, y1, z1) = F(2, 4, 7) = 247
F(x3, y1, z1) = F(3, 4, 7) = 347

```

```

F(x1, y2, z1) = F(1, 5, 7) = 157
F(x2, y2, z1) = F(2, 5, 7) = 257
F(x3, y2, z1) = F(3, 5, 7) = 357

F(x1, y3, z1) = F(1, 6, 7) = 167
F(x2, y3, z1) = F(2, 6, 7) = 267
F(x3, y3, z1) = F(3, 6, 7) = 367

F(x1, y1, z2) = F(1, 4, 8) = 148
F(x2, y1, z2) = F(2, 4, 8) = 248
F(x3, y1, z2) = F(3, 4, 8) = 348

F(x1, y2, z2) = F(1, 5, 8) = 158
F(x2, y2, z2) = F(2, 5, 8) = 258
F(x3, y2, z2) = F(3, 5, 8) = 358

F(x1, y3, z2) = F(1, 6, 8) = 168
F(x2, y3, z2) = F(2, 6, 8) = 268
F(x3, y3, z2) = F(3, 6, 8) = 368

F(x1, y1, z3) = F(1, 4, 9) = 149
F(x2, y1, z3) = F(2, 4, 9) = 249
F(x3, y1, z3) = F(3, 4, 9) = 349

F(x1, y2, z3) = F(1, 5, 9) = 159
F(x2, y2, z3) = F(2, 5, 9) = 259
F(x3, y2, z3) = F(3, 5, 9) = 359

F(x1, y3, z3) = F(1, 6, 9) = 169
F(x2, y3, z3) = F(2, 6, 9) = 269
F(x3, y3, z3) = F(3, 6, 9) = 369

```

If all function arguments match one of the specified grid points, that is, each argument matches one of the table values for the corresponding coordinate, the result of the function is the value specified in the table for that set of coordinates. When the function arguments fall between two specified values, the function must be interpolated. In order to evaluate the N-dimensional function  $F(x_1, x_2, \dots, x_L, x_M, x_N)$  where  $M=N-1$  and  $L=N-2$ , we first find the bounding box, or the grid cell containing the specified N-dimensional point  $(x_1, x_2, \dots, x_L, x_M, x_N)$ . For each coordinate  $x_i$ , we find the closest lower and upper values,  $x_{il}$  and  $x_{iu}$  respectively, such that  $x_{il} \leq x_i < x_{iu}$ . Then the function value is calculated by successive linear interpolation in each dimension starting from the last one:

```

F(x1, x2, ..., xL, xM, xN) = F(x1, x2, ..., xL, xM, xNl) +
(xN - xNl)/(xNu - xNl)*( F(x1, x2, ..., xL, xM, xNu) -
F(x1, x2, ..., xL, xM, xNl) )

F(x1, x2, ..., xL, xM, xNl) = F(x1, x2, ..., xL, xMl, xNl) +
(xM - xMl)/(xMu - xMl)*( F(x1, x2, ..., xL, xMu, xNl) -
F(x1, x2, ..., xL, xMl, xNl) )

F(x1, x2, ..., xL, xM, xNu) = F(x1, x2, ..., xL, xMl, xNu) +
(xM - xMl)/(xMu - xMl)*( F(x1, x2, ..., xL, xMu, xNu) -
F(x1, x2, ..., xL, xMl, xNu) )

F(x1, x2, ..., xL, xMl, xNl) = F(x1, x2, ..., xLl, xMl, xNl) +

```

$$(xL - xL1)/(xLu - xL1) * ( F(x1, x2, \dots, xLu, xM1, xN1) - F(x1, x2, \dots, xL1, xM1, xN1) )$$

and so on. For example, a 4-dimensional function  $F(w, x, y, z)$  where the point  $(w, x, y, z)$  falls within the grid cell  $((w1, w2), (x1, x2), (y1, y2), (z1, z2))$ , is evaluated as follows:

$$F(w, x, y, z) = F(w, x, y, z1) + (z - z1)/(z2 - z1) * ( F(w, x, y, z2) - F(w, x, y, z1) )$$

The values for functions  $F(w, x, y, z1)$  and  $F(w, x, y, z2)$  are then found through interpolation by finding bounding values on the  $wx$  planes:

$$F(w, x, y, z1) = F(w, x, y1, z1) + (y - y1)/(y2 - y1) * ( F(w, x, y2, z1) - F(w, x, y1, z1) )$$

$$F(w, x, y, z2) = F(w, x, y1, z2) + (y - y1)/(y2 - y1) * ( F(w, x, y2, z2) - F(w, x, y1, z2) )$$

The values for  $F(w, x, y1, z1)$  and  $F(w, x, y2, z1)$  are interpolated to find the value for the first equation above and the values for  $F(w, x, y1, z2)$  and  $F(w, x, y2, z2)$  are interpolated to find the values for second equation:

$$F(w, x, y1, z1) = F(w, x1, y1, z1) + (x - x1)/(x2 - x1) * ( F(w, x2, y1, z1) - F(w, x1, y1, z1) )$$

$$F(w, x, y2, z1) = F(w, x1, y2, z1) + (x - x1)/(x2 - x1) * ( F(w, x2, y2, z1) - F(w, x1, y2, z1) )$$

$$F(w, x, y1, z2) = F(w, x1, y1, z2) + (x - x1)/(x2 - x1) * ( F(w, x2, y1, z2) - F(w, x1, y1, z2) )$$

$$F(w, x, y2, z2) = F(w, x1, y2, z2) + (x - x1)/(x2 - x1) * ( F(w, x2, y2, z2) - F(w, x1, y2, z2) )$$

Finally, when precise coordinates are described by the functions, the values are found in the table to obtain a value.

$$F(w, x1, y1, z1) = F(w1, x1, y1, z1) + (w - w1)/(w2 - w1) * ( F(w2, x1, y1, z1) - F(w1, x1, y1, z1) )$$

$$F(w, x2, y1, z1) = F(w1, x2, y1, z1) + (w - w1)/(w2 - w1) * ( F(w2, x2, y1, z1) - F(w1, x2, y1, z1) )$$

$$F(w, x1, y2, z1) = F(w1, x1, y2, z1) + (w - w1)/(w2 - w1) * ( F(w2, x1, y2, z1) - F(w1, x1, y2, z1) )$$

$$F(w, x2, y2, z1) = F(w1, x2, y2, z1) + (w - w1)/(w2 - w1) * ( F(w2, x2, y2, z1) - F(w1, x2, y2, z1) )$$

$$F(w, x1, y1, z2) = F(w1, x1, y1, z2) + (w - w1)/(w2 - w1) * ( F(w2, x1, y1, z2) - F(w1, x1, y1, z2) )$$

$$F(w, x2, y1, z2) = F(w1, x2, y1, z2) + (w - w1)/(w2 - w1) * ( F(w2, x2, y1, z2) - F(w1, x2, y1, z2) )$$

$$F(w, x1, y2, z2) = F(w1, x1, y2, z2) + (w - w1)/(w2 - w1) * ( F(w2, x1, y2, z2) - F(w1, x1, y2, z2) )$$

$$F(w, x2, y2, z2) = F(w1, x2, y2, z2) + (w - w1)/(w2 - w1) * ( F(w2, x2, y2, z2) - F(w1, x2, y2, z2) )$$

If the specified N-dimensional point falls outside of the table in at least one dimension, the expression evaluation fails and the following warning is issued:

WARNING: Expression(s) with at least one evaluation failure:

In rule file <fila name> at line <line> (in operation DFM PROPERTY):  
Coordinate out-of-range when evaluating a MATRIX

The following syntax checks are performed when a DFM Function **MATRIX** statement is parsed, and error messages are issued if any of them are violated:

- There must be at least one numeric value for each variable defined.
- For each variable, its values must be sorted in ascending order.
- The number of output values must equal the product of the number of input values.
- The number of columns must equal the number of values provided for the highest dimension.
- The number of rows must equal the product of the number of values found for the lowest dimension up to the next-to-last dimension.
- Neither the SPLINE or LINEAR keywords are allowed for multi-dimensional tables.
- **TVF\_NUMBER\_FUNCTION('procedure\_name ','library\_name')**

Used to call a TVF procedure from a DFM Function statement. The *procedure\_name* is the name of the Tcl procedure, and the *library\_name* is the name of the TVF library containing the procedure.

TVF procedures called from a DFM Function statement may only return a numeric value. They cannot call other DFM functions nor built-in functions within Calibre. Each TVF procedure can be used by only one DFM Function. DFM Function statements that call TVF functions may be used only in DFM Property and DFM Analyze operations.

The arguments passed to a DFM function are not directly passed to a Tcl script. Instead, an interface is provided such that a Tcl script can access each argument. The Tcl procedure must have knowledge of the arguments, their types and order to retrieve the values passed to the DFM function. Tcl procedures referenced by DFM Function statements must not contain default values. For example, the following code is not allowed, because the ng\_ and vg\_ arguments have default values.

```
// NOT ALLOWED
DFM FUNCTION [ PARALLEL_COUNT(NETID N1, NETID N2, VNETID V1, VNETID
V2, NETID NB, VNETID VB) TVF_NUM_FUN(parallel_count, devlib) ]

TVF FUNCTION devlib [
proc parallel_count {n1_ n2_ v1_ v2_ nb_ vb_ { ng_ "" } { vg_ "" } }
{
    ...
}
]
```

The arguments are passed to the Tcl script as command objects which provide access to the arguments passed to the DFM Function (one command object per argument). There are separate command objects for NUMBER, STRING, VECTOR and VSTRING arguments.

To access a number, a string or a netID, the corresponding command object must be evaluated. For example:

```
DFM FUNCTION [ F( NUMBER N, STRING S, NETID N_ID )
    TVF_NUM_FUN( myproc, mylib ) ]
```

```
TVF FUNCTION mylib /*  
proc myproc { n_ s_ n_id_ } {  
    # Access number and string arguments  
    set n [$n_]  
    set s [$s_]  
    set n_id [$n_id_]  
    return $n  
}  
*/]
```

The command object that provides access to vector, string vector, and vnetID arguments supports several commands:

- *tuple\_count* — the number of tuples.
- *tuple\_size* — the tuple size.
- *value i j* — accesses the j-th value in the i-th tuple.

For example:

```
DFM FUNCTION [ F( VECTOR V, VNETID NV )  
    TVF_NUM_FUN( myproc, mylib ) ]  
TVF FUNCTION mylib /*  
proc myproc { v_ nv_ } {  
    # Access vector arguments  
    set count [$v_ tuple_count]  
    set size [$v_ tuple_size]  
    set x [$v_ value 0 0]  
  
    # Access vnetid arguments  
    set count [$nv_ tuple_count]  
    set size [$nv_ tuple_size]  
    set x [$nv_ value 0 0]  
  
    return $x  
}  
*/]
```

Another example:

```
DFM FUNCTION [ F_VSTRCNT(VSTRING vs) TVF_NUMBER_FUNCTION("str_find",  
    "strfunc") ]  
  
TVF FUNCTION strfunc /*  
proc str_find {get_vstr} {  
    set count [$get_vstr tuple_count]  
    for {set x 0} { $x < $count } {incr x} {  
        set strval [$get_vstr value $x 0]  
        set ck [string compare $strval "pwr"]  
        if {$ck == 0} {  
            return 1  
        }  
    }  
    return 0  
}  
*/]
```

```
P1 = DFM READ M1 PROPERTY VSTRING "key"
MP1 = DFM PROPERTY MERGE M1 P1 [fstr = F_VSTRCNT(SVPROPERTY(P1,
"key"))] > 0
```

Note that indices are zero-based.

The arguments are passed to the TVF procedure in the order specified in the DFM Function statement. The rule compiler verifies that a DFM Function is called with the arguments of the correct type (NUMBER, STRING, VECTOR, or VSTRING). However, there is no compile-time verification that the TVF script accesses the arguments correctly. All TVF errors, including invalid use of the command objects and out-of-range indices, are detected at run-time. If a run-time error is detected, an error message is reported in the transcript and the DFM expression that called the offending DFM Function fails to evaluate.

## Description

Creates a user-defined function that can be used in expressions for [DFM Analyze](#), [DFM Property](#) or other [DFM Function](#) definitions, anywhere a math function (such as `exp()`) can be used.

The validity of a DFM Function is defined in part by the context in which the function is used. For example:

- A measurement function used within a DFM Fill expression cannot contain EC\* or EW.
- A measurement function used within an expression that is evaluated using SUM or PROD cannot contain calls to measurement functions that are not supported with SUM or PROD, such as AREA(), AREA(*layer*), PERIMETER(), PERIMETER(*layer*), LENGTH(*layer*), or COUNT(*layer*).

Note that DFM Function statements are parsed only when they are used in an expression that itself must be parsed, such as those in DFM Analyze or DFM Property commands, or in another DFM Function specification. Therefore, DFM Function statements that are defined in an SVRF file but not referenced are not evaluated for either syntactical correctness or validity.

## Examples

### Example1

The following statement defines a function F, which takes one numeric argument. The function is computed by bicubic interpolation of the table, in the range between 0 and 1.

```
DFM FUNCTION [ F(NUMBER X) TABLE SPLINE
{ 0 0 0.00 0.1 0.10 0.2 0.20 0.3 0.29 0.4 0.38 0.5 0.46
 0.6 0.54 0.7 0.60 0.8 0.66 0.9 0.72 1.0 0.76 } ]
```

Used in DFM Property expressions:

```
DFM PROPERTY M1 [ A2 = F(AREA(M1)) ]
```

**Example 2**

The following statement defines the hyperbolic tangent function:

```
DFM FUNCTION [ TANH(NUMBER X)
    (EXP(X)-EXP(-X)) / (EXP(X)+EXP(-X))
]
```

**Example 3**

The following statement uses the hyperbolic tangent function when the value of the argument is not too large by absolute value, otherwise the asymptotic value of the hyperbolic tangent is returned:

```
DFM FUNCTION [ TANH1(NUMBER X)
    (X < 8) ? (x > -8) ? TANH(X) : -1 : 1
]
```

---

**Note**

DFM Function functions cannot call each other recursively, regardless of whether the recursion is finite or infinite.

---

**Example 4**

The following statement defines a function P, which requires three parameters and returns a scaled property value:

```
DFM FUNCTION [ P(LAYER L, STRING PROPNAM, NUMBER SCALE)
    PROPERTY(L, PROPNAM) * SCALE ]
```

For this function to be valid, the value of PROPNAM must be the name of a property defined using the [DFM Property](#) operation.

**Example 5**

The following function specifies a 3-dimension table on a 2x3x4 grid:

```
DFM FUNCTION [ dfm_func( number x1, number x2, number x3 )
    MATRIX TRUNCATE
        { 0, 1 }
        { 2, 3, 4 }
        { 5, 6, 7, 8 }
        :::
        { 025 125 }
        { 035 135 }
        { 045 145 }
        { 026 126 }
        { 036 136 }
        { 046 146 }
        { 027 127 }
        { 037 137 }
        { 047 147 }
        { 028 128 }
        { 038 138 }
        { 048 148 }
]
```

This table specifies the following function values:

```

F(x1, y1, z1) = F(0, 2, 5) = 025
F(x2, y1, z1) = F(1, 2, 5) = 125

F(x1, y2, z1) = F(0, 3, 5) = 035
F(x2, y2, z1) = F(1, 3, 5) = 135

F(x1, y3, z1) = F(0, 4, 5) = 045
F(x2, y3, z1) = F(1, 4, 5) = 145

F(x1, y1, z2) = F(0, 2, 6) = 026
F(x2, y1, z2) = F(1, 2, 6) = 126

F(x1, y2, z2) = F(0, 3, 6) = 036
F(x2, y2, z2) = F(1, 3, 6) = 136

F(x1, y3, z2) = F(0, 4, 6) = 046
F(x2, y3, z2) = F(1, 4, 6) = 146

F(x1, y1, z3) = F(0, 4, 6) = 027
F(x2, y1, z3) = F(1, 4, 6) = 127

F(x1, y2, z3) = F(0, 4, 6) = 037
F(x2, y2, z3) = F(1, 4, 6) = 137

F(x1, y2, z3) = F(0, 4, 6) = 047
F(x2, y2, z3) = F(1, 4, 6) = 147

F(x1, y1, z4) = F(0, 4, 6) = 028
F(x2, y1, z4) = F(1, 4, 6) = 128

F(x1, y2, z4) = F(0, 4, 6) = 038
F(x2, y2, z4) = F(1, 4, 6) = 138

F(x1, y3, z4) = F(0, 4, 6) = 048
F(x2, y3, z4) = F(1, 4, 6) = 148

```

### Example 6

This example uses TVF\_NUMber\_FUNction to call a TVF procedure:

```

DFM FUNCTION[ get_stat( STRING S, NUMBER N, VECTOR V )
    TVF_NUM_FUNC( get_stat_proc, stat_lib ) ]

TVF FUNCTION stat_lib /* 
    proc get_stat_proc { type_ limit_ vector_ } {
        set stat_type [$type_]
        set limit [$limit_]
        set tuple_count [$vector_ tuple_count]
        set tuple_size [$vector_ tuple_size]
        if { expr {$limit != 0 && $limit < $tuple_count} == 1 } {
            set tuple_count $limit
        }
        if { $stat_type == "count" } {
            set rval $tuple_count
        } else {
            set sum = 0.0
            for { set i 0 } { $i < [ expr {$tuple_count - 1}] } {incr i} {

```

```
for {set j 0} { $j < [ expr {$tuple_size - 1} ] } {incr j} {
    set val [ $vector_ value $i $j ]
    set sum [ expr {$sum + $val} ]
}
set rval [expr {$sum/($tuple_count * $tuple_size) } ]
}
return $rval
}
*/]
```

The first argument is a string that determines if the tuple count or the average value of all the tuple values is returned. If its value is “count”, the tuple count is returned, otherwise the average is returned.

The second argument limits the number of tuples to consider. If the value is not 0, and the tuple count is less than the actual count, the number of tuples used is limited by the argument value.

The third parameter is the actual vector of numeric values.

# DFM GCA

Layer operation

**DFM GCA** *layer [layer ...]*  
 {**SHORT** [POLYGONAL] [{EXCLUDE FAULTS BETWEEN *layerA* '&' *layerB*} ...] |  
**OPEN** [{PORTSLAYER *port\_layer1* [GROUPED BY *group\_layer1*] } [PORTSLAYER  
*port\_layer2* GROUPED BY *group\_layer2*]}]  
**R** *r* [D *depth*] [METRIC {SQUARE | OCTAGONAL}]  
[INSIDE OF {EXTENT | *x1 y1 x2 y2* | LAYER *markerlayer1*}]

Used only in Calibre YieldAnalyzer applications.

## Summary

Outputs a derived polygon layer that represents areas susceptible to producing open or short circuits in the presence of random particles of a given radius. GCA stands for Geometric Critical Area.

## Parameters

- ***layer*** [*layer ...*]

A required argument that specifies one or more polygon (original or derived) input layers. Multiple input layers are allowed only if there is no interaction between the layers; that is, none of the shapes on one input layer touch or overlap any of the shapes on another input layer. All input layers must appear in Connect statements.

- **SHORT** [POLYGONAL] [{EXCLUDE FAULTS BETWEEN *layerA* '&' *layerB*} ...] |  
**OPEN** [{PORTSLAYER *port\_layer1* [GROUPED BY *group\_layer1*] } [PORTSLAYER  
*port\_layer2* GROUPED BY *group\_layer2*]}]

A required keyword set that specifies the type of failure for which critical area is to be calculated, which must be one of the following:

- **SHORT** — calculates shorts between different nets on the input layer.
- **OPEN** — calculates opens for nets on the input layer.

By default, only shorts between shapes belonging to different nets are considered faults. When the POLYGONAL keyword is specified, shorts between any two shapes are considered faults.

The EXCLUDE FAULTS BETWEEN keyword set, which may be specified multiple times, instructs the operation not to consider defects that short a *layerA* shape with a *layerB* shape. This keyword set is valid only when multiple input layers are specified. The arguments *layerA* and *layerB* can be the same layer, and can be specified in either order without affecting the output. The OCTAGONAL metric is not valid with this keyword set.

By default, any defect that covers two opposite edges of a shape is considered an open fault. The PORTSLAYER ... GROUPED BY keyword set modifies this behavior such that open faults occur only where a defect breaks a path between two different port groups.

The PORTSLAYER keyword set has three forms:

- PORTSLAYER *port\_layer1* — each shape on *port\_layer1* is a separate port group.
- PORTSLAYER *port\_layer1* GROUPED BY *group\_layer1* — each port group is defined by the set of all *port\_layer* shapes that intersect with a *group\_layer1* shape.
- PORTSLAYER *port\_layer1* GROUPED BY *group\_layer1* PORTSLAYER *port\_layer2* GROUPED BY *group\_layer2* — defines two port groups and two group layers; each port group is defined by the set of all port layer shapes that intersect with a corresponding group layer shape. This form is useful for assigning port groups to via layers that lie both above and below a metal layer.

- **R r**

A required keyword and argument that specifies a defect radius value in user units.

- **D depth**

An optional keyword and argument pair that specifies the minimal overlap depth with a layout shape for a short or open to be considered a fault. The behavior of this keyword is identical to that of the D keyword in the [DFM Critical Area](#) operation.

- **METRIC {SQUARE | OCTAGONAL}**

An optional keyword set that specifies the metric to use when determining edges within the defect radius (the default is SQUARE).

- **INSIDE OF {EXTENT | *x1 y1 x2 y2* | LAYER *markerlayer1*}**

An optional keyword and value set that defines the analysis region. By default, the analysis region is the layout extent. You can define a different analysis region by either specifying coordinates (*x1 y1 x2 y2*) or by specifying a marker layer with the LAYER keyword. If LAYER is used, the analysis region is the area inside of the specified marker layer.

## Description

Outputs a derived polygon layer that represents areas susceptible to producing open or short circuits in the presence of random particles of a given radius. This operation produces flat output and is intended primarily for visualization of the [DFM CAF](#) operation. It is not intended for use in production flows.

A number of environment variables can be used to control the behavior of this operation:

- **CALIBRE\_DFM\_OPEN\_GCA\_DEANGLING\_PRECISION** — controls the precision (in user units) of the orthogonalization of critical area for OPEN faults. The default is 0.01. Smaller values can impact performance on non-rectilinear designs.
- **CALIBRE\_DFM\_CAA\_MAX\_EDGES\_PER\_TILE** — specifies the upper bound for the partition size as the average flat edge count per tile. The default value is 1,000,000. If this variable is set to a non-positive value, then every cell is processed as one tile. In order to enable the MT/MTflex parallelism of the DFM CAF operation this variable must be set to a reasonable value.

- `CALIBRE_DFM_CAA_MIN_EDGES_PER_TILE` — specifies the lower bound for the partition size as the average flat edge count per tile. The default value is 10,000. Low scalability of parallel processing can result when sparse layers are processed in fewer tiles. In these cases, the operation uses tiles of smaller size (but not less than the specified lower bound) so that all threads are utilized.

## Examples

```
// Outputs geometric short critical area for layer M1 for defect
// radius 0.2
M1_SHORT_GCA = DFM GCA M1 SHORT R 0.2
```

## DFM Grow

Layer operation

**DFM GROW** *layer*

**TARGETAREA** *value*

**MINWIDTH** *value*

**SPACE** *value*

**STEP** *value*

**ITERATIONS** *count*

Used only in Calibre YieldEnhancer applications.

### Summary

Layer operation used to expand polygons uniformly.

This operation creates a derived polygon layer by uniformly expanding polygons on *layer*. The algorithm attempts to enlarge each polygon on the input layer until the polygon's area reaches the value specified by the **TARGETAREA** keyword. The algorithm prohibits notch, spacing, and width violations, as governed by the values associated with secondary keywords **SPACE** and **MINWIDTH**. The **ITERATIONS** parameter globally controls the number of times the expansion **STEP** is applied. Polygons are expanded until they either meet the **TARGETAREA** condition, or until the **ITERATIONS** count is met. The **MINWIDTH** and **SPACE** conditions are enforced throughout the expansion and are never violated.

This operation is a non-node-preserving layer constructor.

### Parameters

- ***layer***

A required original or derived polygon layer. Polygons from this layer that meet the criteria of this operation are expanded.

- **TARGETAREA** *value*

A required keyword and floating-point number pair in square user units that specify the desired minimum area for each polygon. Polygons on the *layer* will either achieve this minimum area condition, or grow by the specified number iterations, whichever comes first.

- **MINWIDTH** *value*

A required keyword and floating-point number pair that specify the minimum width, in user units, a polygon must have *before* expansion occurs. At least one edge of the polygon must be greater than *value* for expansion to occur.

- **SPACE** *value*

A required keyword and floating-point number pair that specify the minimum edge spacing, in user units, to be enforced during polygon expansion. This spacing is always enforced between expanded *layer* edges, even if this prevents polygon expansion.

- **STEP value**

A required keyword and floating-point number pair that specify the increment, in user units, by which to expand *layer* polygons. The expansions occur in steps until one of the following occurs:

- the edge spacings approach the **SPACE value**
- the **TARGETAREA** is met
- the global **ITERATIONS** count is met

- **ITERATIONS count**

A required keyword and positive integer pair that specify the number of global expansion steps the algorithm applies. The **STEP** value is used in each iteration. For each polygon that can be expanded, either this count is achieved, or the area condition is met, whichever comes first. The *count* should be less than 60.

## Description

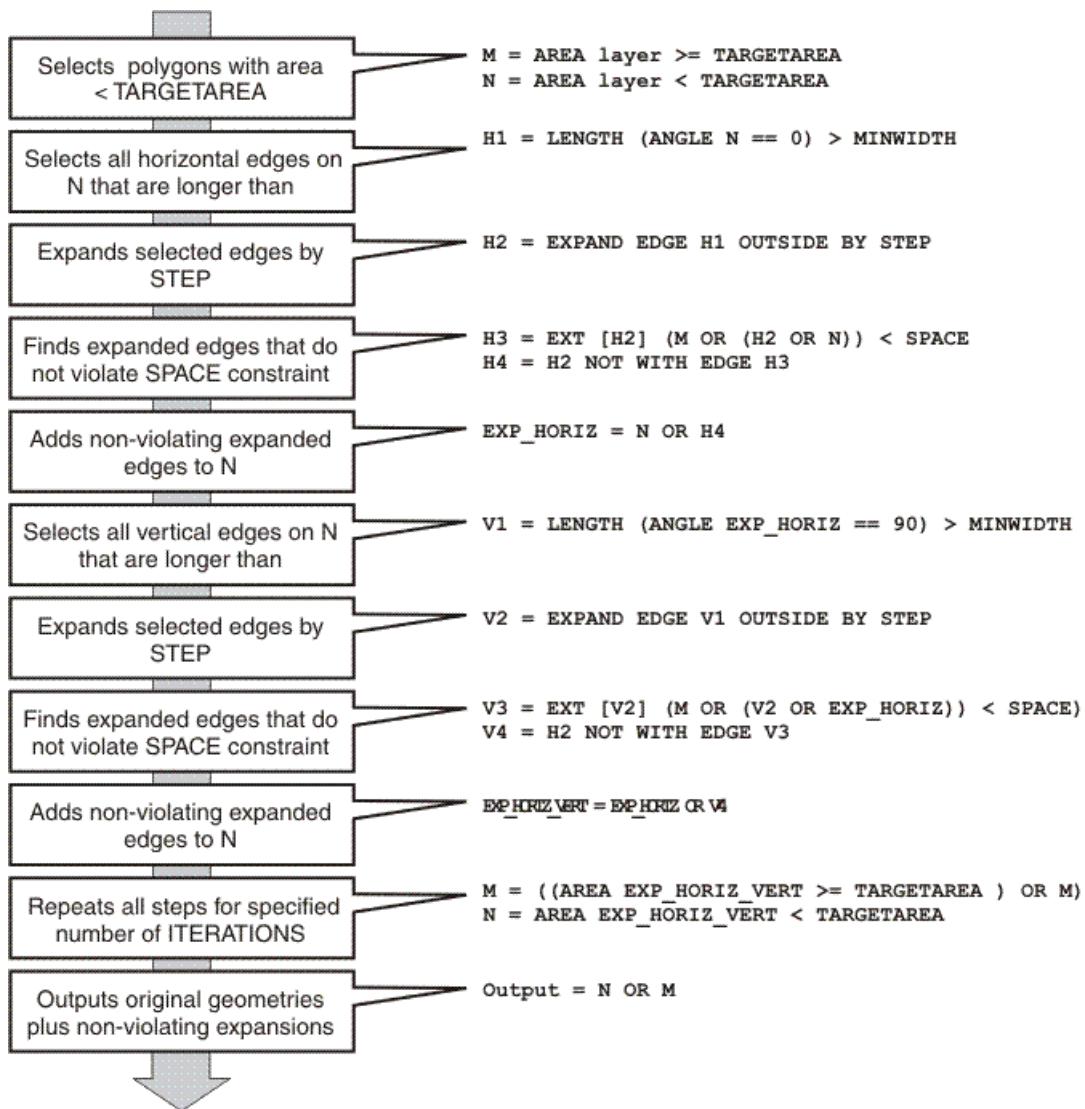
Expands polygons on the input layer (which must be original or derived polygon type) until a minimum area condition is met, or until a specified number of growth iterations is met. DFM Grow creates a derived polygon layer based upon alternate horizontal and vertical [Expand Edge](#) operations. The solutions that DFM Grow finds may be different in flat versus hierarchical runs.

During expansion, the following principles are applicable:

- Expanded polygon edges are longer than the original polygon edges.
- Expanded polygon edge spacing never violates the **SPACE** condition. The following layer operation is used to check spacing:

**EXT [x] y < SPACE**

- Expanded polygons never violate the **MINWIDTH** condition. This is done by checking the lengths of polygon edges to see if they meet this condition.

**Figure 4-56. How DFM Grow Processes Your Data**

# DFM Histogram

Layer operation

```
DFM HISTOGRAM file_name layer_name [ layer_name ... ]
PROPERTY property_name
[ MIN { min_value | AUTO } ] [ MAX { max_value | AUTO } ]
[ BINS num_bins | STEP step_size | AVERAGE average ]
[ BY CELL [ NOPSEUDO ] [ NOEMPTY ] [ ACCUMULATE ] ]
[COMMENT “comment_string”] ...
```

Used only in Calibre YieldAnalyzer applications.

## Summary

DFM Histogram allows you to analyze the distribution of DFM property values in different cells or for the entire chip. This command computes a histogram of property values and prints it out as a text file.

## Parameters

- ***file\_name***

A required argument defining the name of the file to which the histogram will be written. Multiple DFM Histogram commands can use the same file, in which case, each new histogram is appended at the end of the file. Refer to [Histogram File](#) for more details on how the file is organized.

- ***layer\_name* ...**

A required argument supplying the name of a derived layer used to compute the distribution of properties. The layer must be a derived layer created using the [DFM Property](#) operation. At least one layer must be specified, and more are allowed.

If multiple layers are specified, the resulting histogram represents the cumulative distribution across all layers.

- **PROPERTY *property\_name***

A required keyword and argument pair defining the property to be represented as a histogram. If the named property is not present on one or more of the input layers, a warning is printed in the transcript and these layers are ignored by the DFM Histogram command. If the named property is not present on any layers, the command does not produce any output.

- **MIN { *min\_value* | AUTO } MAX { *max\_value* | AUTO }**

An optional keyword and argument set that defines the lower and upper boundaries of the range of the property values represented by the histogram. The argument AUTO, which is the default, instructs Calibre to automatically compute the lower and upper limits based on the property values. These limits are then used to determine the centers of the leftmost and rightmost bins.

If a property value falls on the border between two bins, it is counted in the bin on the right side of the border.

If the STEP keyword is specified, the *step\_size* is used and the bin count is changed appropriately. If both lower and upper limits are computed automatically, it is generally not possible to center both the leftmost and the rightmost bins on the extreme property values. In this case, the leftmost bin is centered on the lower limit and the rightmost bin is guaranteed to enclose the rightmost property value. If only one limit is computed automatically, then that limit is used as a center of the corresponding bin, and the other (explicitly specified) limit remains unchanged. If the BINS keyword is specified, the *num\_bins* value is used and the step is changed appropriately.

For example, assume that the property values are 1, 2, 3, 4, and 5, and the DFM Histogram operation specifies MIN AUTO MAX AUTO STEP 1. Five bins are created with ranges defined as follows: 0.5-1.5, 1.5-2.5, 2.5-3.5, 3.5-4.5, 4.5-5.5.

- **BINS *num\_bins***

An optional keyword and argument pair defining a fixed number of bins for the histogram. This keyword cannot be used with STEP or AVERAGE. If no binning options are specified, the default is to use 10 bins.

- **STEP *step\_size***

An optional keyword and argument pair defining step size for all the bins for the histogram. The step size is the differences between the lowest value in one bin and the lowest value in the adjacent bins.

This keyword cannot be used with BINS or AVERAGE. If no binning options are specified, the default is to use 10 bins.

- **AVERAGE *average***

An optional keyword and argument pair instructing the operation to automatically calculate the sizes of bins such that the average number of results in one bin equals the value specified with the AVERAGE option.

This keyword cannot be used with STEP or BINS. If no binning options are specified, the default is to use 10 bins.

- **BY CELL**

An optional keyword instructing the DFM Histogram operation to generate a separate histogram for each cell in the design. All histograms have the same range and bin count. By default, the DFM Histogram command computes the distribution for top-level cell.

- **NOPSEUDO**

An optional keyword, used only with BY CELL, that instructs the DFM Histogram operation to skip pseudo (Calibre-created) cells. The results from these cells are counted in the design cells containing the pseudo cell; in other words, the results are flattened up through pseudo-cells.

- NOEMPTY

An optional keyword, used only with BY CELL, that instructs the DFM Histogram operation to skip cells that have no results from the output. Without this option, a histogram with all zero values is printed for such cells.

- ACCUMULATE

An optional keyword and argument, used only with BY CELL, to control the way results are collected in each cell:

- Without ACCUMULATE, the histogram reports the results in the top level of the specified cell itself (and, possibly, in pseudo cells contained in the cell).
- With ACCUMULATE, the histogram reports the results all levels of the specified cell. That is, the cell is considered flat, allowing the results from all cells referenced within the specified cell to be counted. ACCUMULATE and NOPSEUDO can be used together.

- COMMENT “*comment\_string*”

An optional keyword and argument used to add comments to the histogram file. May be specified multiple times. Each comment adds one line of check text to the histogram file, directly following the first two header lines.

The *comment\_string* must be a single string, so comments containing spaces or special characters must be enclosed in double quotes (“ ”). While required only for comments containing spaces, double quotes are recommended for all comments.

## Examples

The following code finds locations where two polygons on layers MET1 or MET2 are closely spaced and computes the distribution of lengths of these areas:

```
MET1S = EXT MET1 < 0.25
MET2S = EXT MET2 < 0.2
MET1SE = DFM PROPERTY MET1S [ EC = EC( MET1S ) ]
MET2SE = DFM PROPERTY MET2S [ EC = EC( MET2S ) ]
CHECK {
    DFM HISTOGRAM "hist1_2h" MET1SE MET2SE PROPERTY "EC" BY CELL
    ACCUMULATE BINS 5
}
```

The following is an example of the histogram data produced by the code:

```
top 1000
PROPERTY EC: MET1SE MET2SE
5 BINS: MIN 0(0) MAX 1.9(0)
CELL cell1(ACCUMULATED)
0.19: 3 0.56391 0.214286
0.57: 3 0.56391 0.214286
0.95: 0 0 0
1.33: 8 1.50376 0.571429
1.71: 0 0 0
5 BINS: MIN 0(0) MAX 1.9(0)
CELL cell2(ACCUMULATED)
```

## DFM Histogram

---

```
0.19: 3 0.657895 0.25
0.57: 5 1.09649 0.416667
0.95: 2 0.438596 0.166667
1.33: 2 0.438596 0.166667
1.71: 0 0 0
5 BINS: MIN 0(0) MAX 1.9(0)
CELL top(ACCUMULATED)
0.19: 20 0.93985 0.357143
0.57: 11 0.516917 0.196429
0.95: 5 0.234962 0.0892857
1.33: 18 0.845865 0.321429
1.71: 2 0.093985 0.0357143
```

## DFM Measure

Layer operation

```
DFM MEASURE layer [REGION] [BY POLYGON [MEASURE ALL]]  

  {measurement_rule [OPPOSITE | OPPOSITE EXTENDED value | SQUARE]  

  [PROJECTING constraint]  

  [measurement_rule ...]}  

  [RDB_specification]
```

Used only in Calibre YieldAnalyzer applications.

### Summary

Layer operation used to perform single-layer, table-based width and spacing checks. You can optionally output the results to a [Check Results RDB](#) for statistical analysis. This operation allows you to define many checks through a single operation.

### Parameters

- ***layer***  
A required original or derived polygon layer. If specified by itself, with no brackets surrounding the layer name, this produces error-directed output, which cannot be used for derived layer input to any other operation (except [DFM Analyze](#)). For edge-directed output (which can be used for deriving layers), enclose the *layer* parameter in brackets [ ]. This is similar to edge-directed output for the [External](#) and [Internal](#) operations.
- **REGION**  
An optional keyword that generates a derived polygon layer instead of a derived error layer. (This output can be used in layer derivations.) It constructs edge projections between the endpoints of selected edges to create polygonal regions; the composite of the selected edges and edge projections is output as derived polygon data. This is similar to polygon-directed output for the EXternal and INTernal operations. This keyword cannot be used simultaneously with the brackets [ ] used to generate derived edge layers.  
  
If you are deriving polygon layers, using the REGION keyword with the Euclidean (default) metric can generate large numbers of skew edges, which must be flattened in hierarchical applications. This is usually wasteful and requires much excess processing time. You can use the OPPOSITE metric to eliminate skew edges, but be aware that you can miss legitimate design rule errors (such as corner-to-corner configurations) when using OPPOSITE.
- **BY POLYGON**  
An optional keyword that specifies for certain *measurement\_rule* types to alter internal heuristics to perform measurements between derived polygons rather than derived edges.

- MEASURE ALL

An optional keyword that specifies to ignore the polygon containment criteria for certain measurements when BY POLYGON is specified. This allows EXternal dimensional check operations to “see through” polygons that ordinarily they would not.

- ***measurement\_rule***

A mandatory rule that specifies the measurements to perform. You must include at least one ***measurement\_rule*** in the rule check. There are two basic types of measurement rules—spacing and width, and both may be specified in the same rule check. Measurement rules have these basic forms:

- **SPACE *space\_constraint* [WIDTH1 *constraint1*] [WIDTH2 *constraint2*]**

**SPACE *space\_constraint*** — Keyword and constraint instructing the operation to measure spacing between edges on the input layer. When specified alone, the SPACE parameter set measures all edges on the input ***layer*** to see if they meet the *space\_constraint*. When specified with either WIDTH1 or WIDTH2, or both, the width of polygons is taken into account before edge spacing measurements are made.

**WIDTH1 *constraint1*** — Optional keyword and constraint that can be specified with **SPACE *space\_constraint***. When this is specified as the only width condition in a spacing rule, edge spacings are tested where *all* of the edges being measured come from polygons having a width that meets *constraint1*. When specified with **WIDTH2**, edge spacings are tested between edges coming from polygons having widths that meet *constraint1* and *constraint2*, respectively.

**WIDTH2 *constraint2*** — Optional keyword and constraint that can be specified with **SPACE *space\_constraint***. When this is specified as the only width condition in a spacing rule, edge spacings are tested where *one* of the edges being measured comes from a polygon having a width that meets *constraint2*. The other edge comes from any polygon on the input layer that lies within the measurement region. When specified with **WIDTH1**, edge spacings are tested between edges coming from polygons having widths that meet *constraint1* and *constraint2*, respectively.

- **WIDTH *width\_constraint***

Optional keyword and constraint that measure the widths of polygons on the input ***layer***. This is different from the **WIDTH1** or **WIDTH2** keywords in that **WIDTH** rules are distinct from **SPACE** rules. All edges on the input layer are tested to see if they meet the *width\_constraint*.

- **OPPOSITE | OPPOSITE EXTENDED *value* | SQUARE**

Optional secondary keywords used with a ***measurement\_rule*** to specify the metric to use for that rule. You can specify a different metric for each ***measurement\_rule***. Only one of the three metrics may be specified in any spacing or width rule. The default metric is Euclidean, and there is no keyword needed for this metric. These metrics are discussed under EXternal and INTernal, and under “Metrics” in the *Calibre Verification User’s Manual*.

- PROJECTING *constraint*

Optional secondary keyword set used with a ***measurement\_rule*** to measure the separation between edges based upon their mutual edge projection. Note that the projections of edges onto other edges (if any) are not the output, rather the results of the ***measurement\_rule*** and associated optional keywords are output if the specified projection exists. Specifying the PROJECTING keyword also internally sets the PARALLEL ONLY filter for measurements. Edge projection and the PARALLEL ONLY filter are discussed under EXternal and INTernal.

You must specify this keyword set for each ***measurement\_rule*** to which it applies.

- *RDB\_specification*

An optional specification that instructs the operation to write results to a DFM [Check Results RDB](#). This RDB is structured such that each measurement performed is reported as a separate RDB rule check. Within the rule check, results are organized based on the [Partitioning](#) you define through the WINDOW and BY CELL keywords.

The syntactical elements for RDB specification are:

```
RDB [ONLY] filename [MAXIMUM {value | ALL}] [NOEMPTY]
  {{[WINDOW {w | x y}]} [STEP {s | x y}]}
  [INSIDE OF {{x1 y1 x2 y2} | EXTENT | LAYER name}]} |
  [BY CELL [NOPSEUDO]]
```

By default, results of all measurement checks performed are combined into a single rule check in the nmDRC results database. In the RDB, each check is reported as a different rule check, with a system-generated rule check name.

When ONLY is specified, results are sent only to the DFM [Check Results RDB](#). No geometric data is sent to the usual nmDRC results database.

Within the *filename*, the [Pattern Substitution](#) facility replaces all occurrences of the pattern “%\_t\_” with the name of the top cell for the design.

## Description

Selects edges that meet the constraints and associated parameters of spacing and width rules specified in the rule check. DFM Measure is similar to the TDDRC operation, except DFM Measure allows RDB specification.

The DFM Measure commands makes it possible for you to perform all checks defined in a width and spacing table through a single operation. In most cases, each row in the table represents a unique measurement rule in the DFM Measure operation. The exact syntax of the measurement rule varies according to the details of the check, as shown in the table below.

**Table 4-9. Using the Correct Measurement Rule Syntax**

Rule Type	Applies...	Use...
Spacing	To all edges, regardless of width	SPACE without WIDTH1 or WIDTH2

**Table 4-9. Using the Correct Measurement Rule Syntax (cont.)**

Rule Type	Applies...	Use...
Spacing	When both polygons fit a specific width condition (one width constraint used for both polygons)	SPACE with WIDTH1
Spacing	When at least one polygon is of a specific width (one width constraint)	SPACE with WIDTH2
Spacing	When one polygon meets one constraint and the other meets a second constraint (two width constraints)	WIDTH1 and WIDTH2
Width	To all polygons	WIDTH ( <i>no SPACE</i> )

By default, the output is a derived error layer, which cannot be used as input for other operations, except [DFM Analyze](#), [DFM RDB](#) or [DFM Property](#). You can configure the operation to generate either edge or polygon data as follows:

To produce derived *edge data*, enclose the input layer in square brackets.

```
edge_data = DFM MEASURE [metal1]
    SPACE < 0.4 WIDTH1 >= 0.05 < .10 WIDTH2 >= 0.05 < 0.10
    SPACE < 0.3 WIDTH1 < 0.05           WIDTH2 >= 0.05 < 0.10
    SPACE < 0.2 WIDTH1 < 0.05
```

To produce derived *polygon data*, include the REGION keyword.

```
polygon_data = DFM MEASURE metal1 REGION
    SPACE < 0.4 WIDTH1 >= 0.05 < .10 WIDTH2 >= 0.05 < 0.10
    SPACE < 0.3 WIDTH1 < 0.05           WIDTH2 >= 0.05 < 0.10
    SPACE < 0.2 WIDTH1 < 0.05
```

## Relating Measurement Rules to Standard SVRF Operations

These are the formal definitions of all possible width and spacing rules. All of these types of measurement rules can appear in any DFM Measure rule check. For each rule shown, the *metric* refers to the OPPOSITE, OPPOSITE EXTENDED *value*, or SQUARE keywords.

- SPACE *space\_constraint* [*metric*] [PROJECTING *constraint*]
   
EXT *layer space\_constraint* [*metric*] [PROJECTING *constraint*] PARALLEL ONLY] ABUT < 90 SINGULAR
- SPACE *space\_constraint* [*metric*] [PROJECTING *constraint*] WIDTH1 *constraint1*
  
X = *layer COINCIDENT EDGE* (*layer WITH WIDTH constraint1*)
   
EXT X *space\_constraint* [*metric*] [PROJECTING *constraint*] PARALLEL ONLY]
   
ABUT < 90

If BY POLYGON [MEASURE ALL] is specified, then the following operations are used:

$X = layer$  WITH WIDTH *constraint1*  
 EXT X *space\_constraint [metric]* [PROJECTING *constraint* PARALLEL ONLY]  
 ABUT < 90 SINGULAR

- SPACE *space\_constraint [metric]* [PROJECTING *constraint*] WIDTH2 *constraint2*

$X = layer$  COINCIDENT EDGE ( $layer$  WITH WIDTH *constraint2*)  
 EXT X  $layer$  *space\_constraint [metric]* [PROJECTING *constraint* PARALLEL  
 ONLY] ABUT < 90

If BY POLYGON [MEASURE ALL] is specified, then the following operations are used:

$X = layer$  WITH WIDTH *constraint2*  
 EXT X *space\_constraint [metric]* [PROJECTING *constraint* PARALLEL ONLY]  
 ABUT < 90 SINGULAR [MEASURE ALL]

- SPACE *space\_constraint [metric]* [PROJECTING *constraint*]  
 WIDTH1 *constraint1* WIDTH2 *constraint2*

$X = layer$  COINCIDENT EDGE ( $layer$  WITH WIDTH *constraint1*)  
 $Y = layer$  COINCIDENT EDGE ( $layer$  WITH WIDTH *constraint2*)  
 EXT X Y *space\_constraint [metric]* [PROJECTING *constraint* PARALLEL  
 ONLY] ABUT < 90

If BY POLYGON [MEASURE ALL] is specified, then the following operations are used:

$X = layer$  WITH WIDTH *constraint1*  
 $Y = layer$  WITH WIDTH *constraint2*  
 EXT X Y *space\_constraint [metric]* [PROJECTING *constraint* PARALLEL  
 ONLY] ABUT < 90 SINGULAR [MEASURE ALL]

- WIDTH *width\_constraint [metric]* [PROJECTING *constraint*]

INT  $layer$  *width\_constraint [metric]* [PROJECTING *constraint* PARALLEL  
 ONLY] ABUT < 90 SINGULAR

## RDB Output

RDB output is optional, but helpful in most cases where a detailed analysis of the results is desirable. For a complete description of the RDB created by this operation, refer to [DFM Check Results RDB](#).

## Examples

### Example1

Suppose you want to check width and spacing conditions for M1 polygons that are known to affect yield. [Table 4-10](#) shows the intervals of various polygon widths and the spacings between two polygons of the corresponding widths.

**Table 4-10. M1 Width and Spacing (microns)**

polygon 1 width interval	polygon 2 width interval	polygon 1-to-polygon 2 spacing constraints
$\geq 0.05 < 0.10$	$\geq 0.05 < 0.10$	$< 0.4$
$< 0.05$	$\geq 0.05 < 0.10$	$< 0.3$
$< 0.05$	any	$< 0.2$

You can write a DFM Measure rule file operation that checks all of these conditions as follows:

```
m1_spacings_and_widths {
    DFM MEASURE M1
    SPACE < 0.4 WIDTH1 >= 0.05 < .10 // table row 1
    SPACE < 0.3 WIDTH1 < 0.05           WIDTH2 >= 0.05 < 0.10 // table row 2
    SPACE < 0.2 WIDTH2 < 0.05           // table row 3
```

### Relating Checks to Table Rows

Specifying a single width condition using the WIDTH1 keyword (which must be specified with a SPACE condition) enforces that width condition on both polygons for which the spacing is checked. That is, if both polygons satisfy WIDTH1, then the spacing measurement is taken between edges coming from each polygon.

The WIDTH2 keyword has different semantics when used as the only width condition. This enforces the width interval on only one polygon from which a spacing measurement is taken. The edge from a second polygon in the measurement region can come from any polygon. This corresponds to the following line in the rule check:

```
SPACE < 0.2 WIDTH2 < 0.05           // table row 3
```

For this condition, the SPACE measurement is taken from any edge that comes from a polygon satisfying WIDTH2 and any other polygon, regardless of its width. This enforces the conditions of row 4 in the table.

If you specify WIDTH1 and WIDTH2, then one polygon must satisfy the WIDTH1 condition, and another polygon must satisfy the WIDTH2 condition. If these conditions are met, then the spacing is checked. This corresponds to the following line in the rule check:

```
SPACE < 0.3 WIDTH1 < 0.05           WIDTH2 >= 0.05 < 0.10 // table row 2
```

If you are only interested in measuring widths over certain intervals (but not spacing), you can specify that in the rule check using the WIDTH keyword. For example, assume you are only

interested in performing the width measurements in the first column of [Table 4-10](#). You could write the measurement rules this way:

```
m1_widths {
    DFM MEASURE M1
    WIDTH >= 0.05 < .10
    WIDTH < 0.05
    RDB rdb_file NOEMPTY BY CELL NOPSEUDO
}
```

Note that the WIDTH intervals do not have to be mutually exclusive, as they are in this example. Also notice the WIDTH keyword is different from WIDTH1 (and WIDTH2). The WIDTH keyword is not used in a spacing rule. This example also has RDB output with results organized by cell, which is discussed in [Check Results RDB](#). RDB output is useful in presenting results in these kinds of checks.

In a similar way, you can do spacing checks, with no consideration of width. For example, you can check the third column in [Table 4-10](#) this way, using mutually exclusive intervals (which is not mandatory, but probably more useful than overlapping intervals in many cases):

```
m1_spacings {
    DFM MEASURE M1
    SPACE > 0.4 <= 0.5
    SPACE > 0.3 <= 0.4
    SPACE > 0.2 <= 0.3
    SPACE <= 0.2
    RDB rdb_file NOEMPTY BY CELL NOPSEUDO
}
```

You can also put the previous two examples together into a single rule check, like this:

```
m1_dfm {
    DFM MEASURE M1
    WIDTH >= 0.05 < .10
    WIDTH < 0.05
    SPACE > 0.4 <= 0.5
    SPACE > 0.3 <= 0.4
    SPACE > 0.2 <= 0.3
    SPACE <= 0.2
    RDB rdb_file NOEMPTY BY CELL NOPSEUDO
}
```

### **Example 2**

The following example is a table-based rule check:

```
m1_spacings_and_widths {
    DFM MEASURE M1
    SPACE < 0.4 WIDTH1 >= 0.05 < .10 WIDTH2 >= 0.05 < 0.10
    SPACE < 0.3 WIDTH1 < 0.05           WIDTH2 >= 0.05 < 0.10
    SPACE < 0.2 WIDTH1 < 0.05
    RDB rdb_file MAXIMUM ALL NOEMPTY WINDOW 100
}
```

The third measurement rule measures spacings on the intervals shown, between polygons that meet the WIDTH1 criteria. The first and second rules measure spacings on the intervals shown between polygons that meet the WIDTH1 and WIDTH2 criteria, respectively.

## DFM Measure

---

The RDB specification calls for all results (MAXIMUM ALL), with no windows output that have no results (NOEMPTY), and a  $100 \times 100$  data capture window.

### Example 3

Consider this design rule:

Metal 1 spacing and width must be at least 0.05 in all cases. Spacing must be at least 0.08 between polygons of width greater than or equal to 0.10 and any other M1 polygons. Spacing must be at least 0.06 between polygons of width greater than or equal to 0.05, but less than 0.10, and any other M1 polygons.

This rule can be checked as follows:

```
m1_table {  
    DFM MEASURE M1  
    SPACE < .05  
    WIDTH < .05  
    SPACE < .08 WIDTH2 >= 0.10  
    SPACE < .06 WIDTH2 >= 0.05 < .10
```

# DFM NARAC

Layer operation

## DFM NARAC layer BY *property*

### Parameters

- *layer*

A required argument that specifies the input layer, which must be a derived polygon layer.

- *property*

A required argument that specifies the name of a numeric-type property on the input layer.

The value of this property defines the accumulated NAR values on the output layer.

### Description

Converts a layer containing DFM properties into a Net Area Ratio ACCumulate (NARAC) layer that can be used by Net Area Ratio operations that require a such a layer, such as [Net Area Ratio Accumulate](#). Geometries on the output layer are an exact copy of those on the input layer. The value of the specified property defines the accumulated Net Area Ratio values on the output layer. Any other DFM properties that are associated with the input layer are ignored and are not copied to the output layer.

This operation provides the reverse in functionality to that of the [NARAC\(\)](#) function.

If the output layer is used as a NARAC layer but the specified property does not exist or is not a numeric-type DFM property, a warning is written to the transcript and an empty output layer is created. If the output layer is not used as a NARAC layer, the existence of the specified property is not checked.

### Examples

Consider the following rule file fragment, which uses the [Net Area Ratio](#) operation:

```
a = m1 AND m2
b = m2 AND m1
CONNECT m1 m2 BY v12
an = NET AREA RATIO a >= 0 [AREA(a)] ACCUMULATE
bn = NET AREA RATIO b >= 0 [AREA(b)] ACCUMULATE
ab = NET AREA RATIO ACCUMULATE an bn >= 0.1 [VALUE(an) + VALUE(bn)]
```

By using DFM Property instead of Net Area Ratio to compute the areas of an and bn, you could rewrite the rules as follows:

```
ap = DFM PROPERTY a [A=AREA(a)]
an = DFM NARAC ap BY "A"
bp = DFM PROPERTY b [A=AREA(b)]
bn = DFM NARAC bp BY "A"
ab2 = NET AREA RATIO ACCUMULATE an bn >= 0.1 [VALUE(an)+VALUE(bn)]
```

Note that there is a difference between how DFM Property and Net Area Ratio handle area calculations. In the first case, the area value stored on each an and bn polygon is the total area for the net containing the polygon, while in the second case it is the area of the polygon itself.

To reproduce the results of Net Area Ratio using DFM Property, the following rules can be used:

```
ap = DFM PROPERTY a ACCUMULATE ONLY BY NET [A=AREA(a)]
an = DFM NARAC ap BY "A"
bp = DFM PROPERTY b ACCUMULATE ONLY BY NET [A=AREA(b)]
bn = DFM NARAC bp BY "A"
ab2 = NET AREA RATIO ACCUMULATE an bn >= 0.1 [VALUE(an)+VALUE(bn)]
```

In certain cases, the desired value cannot be computed by Net Area Ratio alone. For example, the following rules calculate area divided by the number of vias on each polygon (instead of area only):

```
ap = DFM PROPERTY a v1 OVERLAP ACCUMULATE ONLY BY NET
    [A=AREA(a)/COUNT(v1)]
an = DFM NARAC ap BY "A"
bp = DFM PROPERTY b v1 OVERLAP ACCUMULATE ONLY BY NET
    [A=AREA(b)/COUNT(v1)]
bn = DFM NARAC bp BY "A"
ab2 = NET AREA RATIO ACCUMULATE an bn >= 0.1 [VALUE(an)+VALUE(bn)]
```

# DFM Property

Layer operation

**DFM PROPERTY** *primary\_layer* [*secondary\_layer* ...]  
 [INTERSECTING] | OVERLAP [ABUT ALSO] [SINGULAR]  
 [MULTI | NOMULTI] [REGION] |  
 NODAL [MULTI | NOMULTI]  
 [CORNER] [CONNECTED | NOT CONNECTED]  
 [[INSIDE OF *x1 y1 x2 y2*] ...] [SPLIT [ALL]]  
 [[ACCUMULATE [ONLY]] BY NET [ONLY] [NOHIER [NOPSEUDO]]]  
 [BY CELL [ONLY] [NOHIER [NOPSEUDO]]]  
*[property\_definition ...]*

where *property\_definition* is

'[ *property\_name* = *expression* ]' [ '!' *constraint*]...

## Summary

Layer constructor that associates user-defined properties to individual objects on the primary layer and creates a derived layer containing only those objects that meet the specified criteria. Unless directed otherwise, the objects on the derived layer are associated with the property values in the form of [DFM Properties](#).

## Parameters

- ***primary\_layer***

A required argument supplying the name of an original or derived layer containing the data to which properties will be added. Data from this layer that meets the criteria of this operation is written to the result layer. This layer can be a polygon layer, an edge layer, or an error layer (edge cluster). Edge and error layers can be unmerged.

If *secondary\_layers* are supplied, the ***primary\_layer*** serves as the driving layer for partitioning layout objects into clusters. Properties are calculated based on all the layout objects in the cluster, but associated only with layout objects on the ***primary\_layer***.

- ***secondary\_layer***

An optional argument used to specify an additional layer for consideration when calculating and assigning properties to layout objects on the ***primary\_layer***. Any layer other than the ***primary\_layer*** that is referenced in any expression for the operation must be specified as a *secondary\_layer*. Any number of *secondary\_layers* is allowed.

When *secondary\_layers* are specified, the operation first creates associations between the layout objects on *secondary\_layers* and those on the ***primary\_layer*** by grouping them together into clusters (refer to [Partitioning by Clusters](#)). Each cluster contains exactly one layout object on the ***primary\_layer***, and a number of layout objects on each *secondary\_layer* that is controlled by the specified clustering mode.

When *secondary\_layers* are supplied, the allowed layer types are governed by the method used for assembling clusters:

- When INTERSECTING is specified, the *primary\_layer* and all *secondary\_layers* must be polygon layers.
- When OVERLAP is specified, there are no constraints on the layer types for either the *primary\_layer* or *secondary\_layers*. That is, any combination of input layers of any type is supported.
- **INTERSECTING** | OVERLAP [ABUT ALSO [SINGULAR]]  
[MULTI | NOMULTI] [REGION] | NODAL [MULTI | NOMULTI]

An optional keyword set used to control how the operation assembles clusters associating data on *secondary\_layers* with data on the *primary\_layer*.

- INTERSECTING — a cluster is defined to be a polygon on the *primary\_layer*, plus all *secondary\_layer* polygons that overlap that polygon. *secondary\_layer* polygons that partially overlap the *primary\_layer* polygon are clipped by the primary layer so that only the portions that overlap the *primary\_layer* are used in the calculations. This clustering method applies only to polygon layers, and is the default behavior.
- OVERLAP — a cluster is defined to be a layout object on the *primary\_layer* plus all layout objects on a *secondary\_layer* with which it has a non-zero area of overlap, as described in [Controlling Cluster Creation with DFM Property](#). The *secondary\_layer* layout objects that partially overlap the *primary\_layer* layout object are not clipped.
  - ABUT ALSO — optional secondary keyword that instructs the operation to include *secondary\_layer* layout objects that abut the *primary\_layer* layout object, as well as those that overlap it.
  - SINGULAR — optional secondary keyword that instructs the operation to consider point interaction of edges when grouping geometries on primary and secondary layers. This keyword must be specified with ABUT ALSO if the primary layer is a polygon layer and one or more secondary layers are also polygon layers. If SINGULAR is specified, then two edges, primary and secondary, that share only a single point that is the end point of one or both edges, are considered interacting. Similarly, a primary layer polygon and an edge that share only a single point are considered interacting if this keyword is specified. The results of this keyword are not valid for polygon-to-polygon interactions.
  - MULTI and NOMULTI — optional secondary keywords that control how the operation treats a layout object on the *secondary\_layer* that overlaps or abuts more than one *primary\_layer* layout object. By default, when a polygon on the *secondary\_layer* can be grouped into several clusters, the operation assigns the layout object to one of these clusters arbitrarily and a warning is issued in the

transcript. Specifying either MULTI or NOMULTI overrides the default behavior as follows:

MULTI — When specified, the operation assigns such a *secondary\_layer* layout object to the cluster for each polygon it overlaps or abuts.

NOMULTI — When specified, the layout object is assigned to one of these clusters arbitrarily but no warning is issued in the transcript. That is, NOMULTI suppresses the ambiguous clustering warning.

- REGION — optional secondary keyword that controls how the operation treats error (edge cluster) data for the purpose of forming clusters. This keyword provides a more flexible interpretation of “overlap”. Without it, only the overlap of edges is considered. When specified, the operation treats error data as if the error clusters formed polygons:

Error clusters with two parallel edges are treated as if they formed trapezoids with vertices located in the endpoints of the edges.

All other error clusters are treated as if they were a polygon formed by their extents.

- NODAL — a cluster is defined to be a layout object on the primary layer plus all layout objects on any secondary layer that have the same node number as the primary layer object. The spatial relations between geometries are ignored. NODAL clustering is supported only for polygon and edge layers, and it requires that all input layers to the DFM Property operation have node numbers.

Note that NODAL clustering is not compatible with the CONNECTED and NOT CONNECTED options, since all geometries grouped together by NODAL clustering must be connected.

- MULTI and NOMULTI — optional secondary keywords that control how the operation treats a layout object on the *secondary\_layer* that has the same node number as more than one *primary\_layer* layout object. By default, the operation arbitrarily assigns such a secondary layout object to one of the clusters for the primary layer objects that have the same node number, and a warning is issued in the transcript. Specifying either MULTI or NOMULTI overrides the default behavior as follows:

MULTI — When specified, the operation assigns such a secondary layer layout object to the cluster for each primary layer object having the same node number.

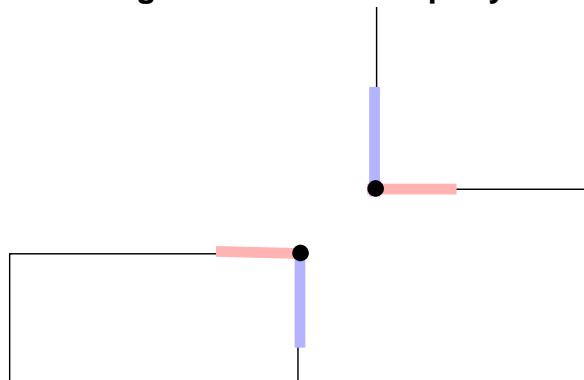
NOMULTI — the operation arbitrarily assigns such a secondary layout object to one of the clusters for the primary layer objects that have the same node number, and no warning is issued in the transcript. That is, NOMULTI suppresses the ambiguous clustering warning.

- CORNER

An optional keyword used to control how the DFM Property operation treats edge clusters (error layers) produced by nmDRC measurements between corners of rectangles aligned to the x and y axes. Because nmDRC measurements operate on edges, measurement between corners produces two errors (two pair of edges) rather than one. This can result in faulty error counts because each corner is counted twice.

Using this keyword instructs the operation to retain only one of the edge clusters produced from nmDRC measurements of corners of rectangles. The operation selects the edge pair to retain that, if converted to a polygon, would have the larger area.

**Figure 4-57. DFM Property CORNER**



This keyword is meaningful only when used with edge clusters, and is valid only with a single layer DFM Property operation. It affects only parallel edge clusters (as shown in [Figure 4-57](#)). Perpendicular edge clusters are unaffected.

- CONNECTED | NOT CONNECTED

Optional keywords that control connectivity filters for clustering geometries on multiple layers. If CONNECTED is specified, only geometries on secondary layers that belong to the same net as the geometry on the primary layer are clustered with that geometry. If NOT CONNECTED is specified, only geometries on secondary layers that do not belong to the same net as the geometry on the primary layer are clustered with that geometry. If either of these keywords is specified, the other geometric requirements for clustering, such as intersection or overlap, must still be met.

The CONNECTED and NOT CONNECTED keywords are supported with INTERSECTING and OVERLAP clustering. These options are supported only in DFM Property operations with at least two input layers and require that all input layers have connectivity information.

- INSIDE OF  $x1\ y1\ x2\ y2$

An optional keyword used to restrict DFM Property to operating on geometries that have at least a partial overlap with the rectangular window specified by the arguments of the INSIDE OF option. This option applies to input layers of any type. An object is said to overlap with the INSIDE OF window if any part of the extent of the object overlaps the window area.

The arguments,  $x1\ y1\ x2\ y2$ , must be specified in chip-level coordinates.

Multiple INSIDE OF options can be specified, defining multiple selection windows. In this case, DFM Property only operates on those geometries that overlap at least one of the windows.

**Note**



When the INSIDE OF option is specified, the DFM Property produces flattened results, with geometries from placements of lower level cells within the window promoted to the top cell. Because of this, this option is intended for use with small windows for detailed analysis of small portions of the chip.

- **SPLIT [ALL]**

Optional keyword that controls merging of edge clusters. By default, primary and secondary edge clusters are merged (unless they contain DFM properties, in which case they are not merged). When SPLIT is specified, secondary layer edge clusters are first merged as usual, then split at all locations where the primary layer has a vertex. The optional ALL keyword extends this behavior to the primary layer. That is, ALL causes the primary layer to be split in all locations where a secondary layer has a vertex. When SPLIT ALL is specified, the behavior of ALL is applied first (the primary layer is split), then the secondary layers are split. See also [Merging of Edge Clusters](#).

- **[[ACCUMULATE [ONLY]] BY NET [ONLY] [NOHIER [NOPSEUDO]]]  
[BY CELL [ONLY] [NOHIER [NOPSEUDO]]]]**

A set of optional arguments used to add cell or net properties to the output layer.

- ACCUMULATE [ONLY] BY NET — ACCUMULATE causes BY NET properties to be stored as polygon properties; the value of a polygon property is the same as the value of the BY NET property for the net to which the polygon belongs. The BY NET properties are also stored unless ONLY is specified. The accumulation of properties is affected by NOHIER and NOPSEUDO in the same way as the BY NET properties themselves.

This option cannot be specified with the BY CELL ONLY or BY NET ONLY options (ACCUMULATE requires creation of polygon properties, and the BY CELL and NET ONLY options prevent creation of these properties). This option may be specified with BY CELL.

- BY NET [ONLY] — calculates the total net property value for each net represented on the output layer. The net property P is the sum<sup>1</sup> of the property P values for all geometries in cell C that are written to the output layer and are associated with the net. By default, net properties are stored on the output layer in addition to the object properties. When ONLY is specified, only net properties are stored on the output layer.

---

1. If the property type is NUMBER then “sum” means arithmetic sum. If the property type is VECTOR then “sum” means concatenation.

- BY CELL [ONLY] — calculates the total cell property value for each cell represented on the output layer. The cell property P is the sum<sup>1</sup> of the property P values for all geometries in cell C that are written to the output layer. By default, cell properties are stored on the output layer in addition to the object properties. When ONLY is specified, only cell properties are stored on the output layer.
- NOHIER — when cell and net properties are calculated, hierarchy is ignored and each cell is evaluated in isolation.
- NOPSEUDO — property values in pseudo cells are promoted up the hierarchy to the closest real cell. This keyword must be specified with NOHIER. If this keyword is not specified, pseudo cells are treated the same as real cells in the design.

For more information, refer to [Cell Properties](#) and [Net Properties](#).

- *property\_definition*

An optional argument set defining the properties to be associated with the geometries on the **primary\_layer**. Multiple instances of *property\_definition* are allowed.

When no properties are specified, the operation functions as a copy, with results filtered based on any INSIDE OF specifications and, for error layers, CORNER.

When one or more properties is specified, results are filtered based on all constraints defined for the properties as well as any INSIDE OF specifications and, for error layers, CORNER.

The format for a *property\_definition* is:

'[ *property\_name* = *expression* ]' [ '['!] *constraint*]...'

where:

- *property\_name*

A required argument defining the property name. The *property\_name* can be any arbitrary string.

Property names “-” and “+” play a special role: these properties are computed and checked against constraints, but not stored on the result layer. These types of properties, referred to as [Nonpersistent DFM Properties](#), are used to filter the input layer when satisfying the constraint is important, but the exact value is not.

Property names need not be unique. Property names that are repeated form expression chains, which can be used to define alternative values in the event that the first expression cannot be evaluated. For more information on expression chains, refer to [Expression Chains](#).

---

1. If the property type is NUMBER then “sum” means arithmetic sum. If the property type is VECTOR then “sum” means concatenation.

- *expression*

An optional expression used to calculate the value of a layer property. The *expression* may involve numbers (including numeric variables), [Unary Operators](#) (+, -, !, ~), [Binary Operators](#) (^, \*, /, +, -), parentheses ( ), variables, [Math Functions](#), [DFM Functions](#), [DFM Properties](#), [Conditional Expressions](#), and most [Measurement Functions](#).

The usual rules of operator precedence apply, and parentheses may be used to establish precedence in the calculation. In addition:

- The *expression* must be enclosed in brackets [ ].
- Division by 0 is defined not to satisfy any constraint.

For more information on expressions, refer to [DFM Expressions](#).

- [!]

An optional constraint option used to negate the constraint that follows. When specified, the DFM Property operation selects all results having property values that do not satisfy the constraint.

The negated constraint is most useful with the interval constraints for which a simple opposite cannot be specified directly. [Expressions that do not evaluate successfully](#) always result in a constraint violation and exclude the polygon from the result, whether the constraint is negated or not.

- *constraint*

An optional DRC-style constraint that applies to the property immediately preceding it. If the constraint is specified, only those geometries on the *primary\_layer* for which the property value satisfies the constraint are included in the result layer. If more than one property has a constraint, a geometry must satisfy all constraints to be included in the result layer.

In addition, if there are any properties of a particular primary polygon for which values cannot be calculated due to [Expressions that Do Not Evaluate Successfully](#), this polygon is never included into the result layer.

## Description

This operation creates a derived layer containing those geometries from the *primary\_layer* that meet the specified criteria. By default, the specified [DFM Properties](#) are associated with each of the geometries on the new layer.

The operation calculates a separate property value for each of the geometries on the *primary\_layer*.

When property calculations involve multiple layers, the operation first creates associations between the geometries on the *primary\_layer* and geometries on any *secondary\_layers* by grouping them together into clusters based on proximity. Each cluster contains exactly one

layout object on the ***primary\_layer***, and a number of layout objects on each ***secondary\_layer*** that is controlled by the specified clustering mode. In this multi-layer case, the operation calculates a separate property value for each cluster, with calculations based on all layout objects in that cluster. The property is associated only with the layout object on the ***primary\_layer***, and only layout objects from the ***primary\_layer*** are written to the resulting derived layer. This operation is a node-preserving layer constructor.

### Promotion

In hierarchical execution, properties are calculated on fully-merged geometries. This will, in general, require that the geometries on primary input layer be promoted. In addition, geometry clusters are assembled across all levels of hierarchy: all geometries in a cluster are promoted as needed so the cluster can be assembled in one cell, and the result is generated in that cell.

The results returned by the DFM Property operation with the INSIDE OF option will produce results in the top level cell (results from placements of lower level cells that belong to the window are promoted to the top cell).

### Concurrency

DFM Property operations are executed concurrently when they share the following characteristics:

- same input layers
- same WINDOW options:
  - INTERSECTING | OVERLAP [ABUT ALSO] [MULTI] [REGION]
  - [CORNER]
  - [[INSIDE OF  $x1\ y1\ x2\ y2$ ] ...]

---

#### Note



You can force multiple operations to run concurrently by supplying a consistent layer list to all DFM Property operations on the same ***primary\_layer***. However, this may have an unexpected effect on promotion if some of the concurrent operations do not use some of the input layers in the expressions (see [Example 2](#)). If this is not desired, do not include unused input layers in the DFM Property operation. This may reduce concurrency, but will ensure that results of one DFM Property operation are not affected by presence of another DFM Property operation.

---

### Merging of Edge Clusters

Hierarchical DFM Property merges edge clusters by default, unless they contain DFM properties, in which case they are not merged. DRC commands that generate edge clusters can split the edges into multiple clusters, particularly on hierarchical boundaries, if part of the cluster can be instantiated in a cell and part cannot. DFM Property merges these edge clusters hierarchically (if they do not contain DFM properties), resulting in “fully merged” clusters: if

two edge clusters, in one cell or different cells, form one cluster when considered flat, these clusters are merged.

### Note

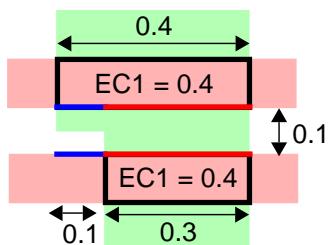


As a result of this merge, measurement functions such as COUNT() are unambiguous when used in DFM Property expressions. This applies only to DFM Property, not to [DFM Analyze](#). However, DFM Analyze can use edge cluster layers that were generated by DFM Property commands and reference properties on the edge clusters. In this case, the input layer to DFM Analyze is already merged by DFM Property and the measurements done by DFM Analyze are unambiguous.

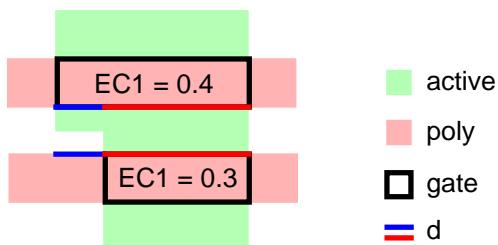
Primary and secondary edge clusters (of any number of edges) that do not contain DFM properties are merged by default. If this is not the desired behavior, you can specify the SPLIT keyword to merge secondary layer edge clusters as usual, then split them at all locations where the primary layer has a vertex (end of an edge, corner of a polygon, or an intersection point of a primary and a secondary edge). For example, in [Figure 4-58](#), by default, the smaller gate polygon includes in the EC() calculation the blue edge pair that touches the polygon in the upper-left corner, since it is merged with the red edge pair coincident with the top edge of the smaller gate polygon. When SPLIT is specified, the blue edge pair is not included in the calculation.

**Figure 4-58. DFM Property SPLIT Behavior**

Default behavior (layer d\_p):



With SPLIT keyword (layer d\_ps):



gate = poly AND active

d = EXT gate poly OPPOSITE <= 0.1

d\_p = DFM PROPERTY gate d OVERLAP ABUT ALSO MULTI [EC1 = EC(d)]

d\_ps = DFM PROPERTY gate d OVERLAP ABUT ALSO SPLIT [EC1 = EC(d)]

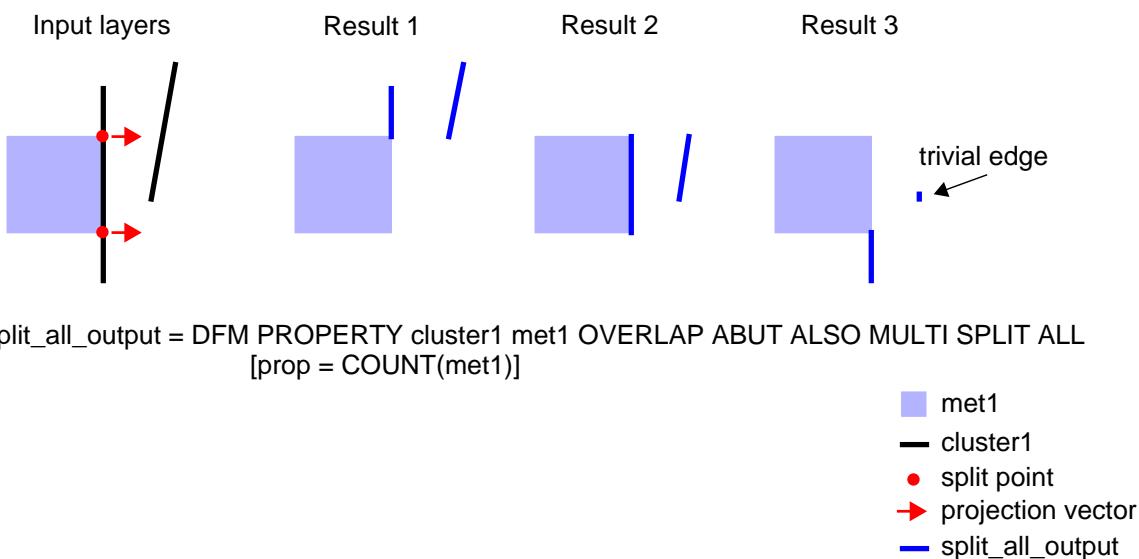
In other words, when SPLIT is specified, the edge pair created by merging layer d is split at the endpoints of the edges on the smaller gate polygon. This creates two edge pairs, one of which touches the upper-left corner of the smaller gate polygon. This edge pair is not included in the EC() calculation.

The SPLIT ALL option, which applies only when the primary layer is a derived error layer, extends the functionality of SPLIT to the primary layer. That is, SPLIT ALL causes the primary layer to be split in all locations where a secondary layer has a vertex (end of an edge, a corner of a polygon, or an intersection point of a primary and a secondary edge). When SPLIT ALL is

specified, the behavior of ALL is applied first (the primary layer is split), then the secondary layers are split. Specifically, SPLIT ALL first splits the clusters of the primary layer at all locations where a secondary layer has a vertex. Points where one edge of a cluster is split are projected onto the other edge to split it as well. After the primary clusters have been split, the SPLIT behavior is then applied to split all secondary layers at all locations where the primary layer has a vertex.

When one edge of an edge cluster is split with the SPLIT or SPLIT ALL option, the other edges of the same cluster, if they exist, are also split. The locations of these subsequent split points are determined by the projections of the original split points. If any cluster edge is axis-aligned, the projection direction is perpendicular to the most prevalent axis-aligned direction: the horizontal or vertical direction with the most edges parallel to it, if there is one, or else the horizontal or vertical direction with the maximum total edge length parallel to it. If no edge is axis-aligned, the projection direction is perpendicular to the cluster edge that is most nearly axis-aligned. If the projection vector does not intersect an edge, a trivial edge is created at the point where the edge is closest to the projection vector (refer to [Figure 4-59](#)).

**Figure 4-59. DFM Property SPLIT and SPLIT ALL Projection**



## Using DFM Property for Device Classification

DFM Property is useful for classifying certain types of advanced devices. See “[Performing Well Enclosure or Stress Effect Calculations with DFM Property](#)” on page 1837. Also see “[DFM Property in ADP Device Recognition](#)” in the *Calibre Verification User’s Manual* for some coding considerations.

## Examples

### Example 1

```
X = DFM PROPERTY A B C [ARATIO = AREA(B)/AREA(C)] [- = AREA(A)] >= 100
```

This command selects all polygons on layer A whose area is no less than 100 in user units. For each of these polygons, the computed property is the ratio of areas of polygons on layers B and C inside the polygon on layer A. This property is stored with every polygon on output layer X and the property name is “ARATIO”. The Area property “-” is used in constraint checking, but is not stored with the layer data.

### Example 2

Consider the following similar lines of code, executed hierarchically:

#### Sample 1

```
X1 = DFM PROPERTY A B [AB = AREA(B)]
X2 = DFM PROPERTY A C [AC = AREA(C)]
```

#### Sample 2

```
X1 = DFM PROPERTY A B C [AB = AREA(B)]
X2 = DFM PROPERTY A B C [AC = AREA(C)]
```

#### Sample 3

```
Y1 = DFM PROPERTY A B C [AB = AREA(B)]
[AC = AREA(C)]
```

Both Sample 1 and Sample 2 create two new layers, X1 annotated with the property AB and X2 annotated with the property AC. However, the commands in Sample 1 are executed separately, while the commands in Sample 2 are executed concurrently. Results are not necessarily the same.

Assume layer C polygons are higher in the design hierarchy than A or B. When performing the operations separately, more promotion would occur when deriving X2 than when deriving X1. When performing the operations concurrently, the system promotes all the geometries needed for both operations, then calculates the properties. Results could differ slightly between X1 derived in Sample 1 and X1 derived in Sample 2, because the latter would have received more promotion.

Both Sample 2 and Sample 3 involve the same hierarchical promotion. However Sample 2 creates two new layers, X1 annotated with the property AB and X2 annotated with the property AC, while Sample 3 creates one new layer, Y1, annotated with two properties, AB and AC.

### Example 3

The following example shows how to annotate gate polygons of a MOS device with the number of contact polygons in the source/drain areas of the device:

```
LAYER POLY 1
LAYER DIFF 2
LAYER CONT 3
SD = DIFF NOT POLY
```

```
GATE = POLY AND DIFF
SD1 = DFM PROPERTY SD CONT [ NC = COUNT(CONT) ]
GATE1 = DFM PROPERTY GATE SD1 OVERLAP ABUT ALSO [ NC = PROPERTY(SD1, NC) ]
```

### Example 4

The following example selects all polygons on MET1 for which the ARATIO property is either less than 1 or greater than 2.

```
DFM PROPERTY MET1 MET2 [ ARATIO = AREA(MET1)/AREA(MET2) ] ! >= 1 <= 2
```

### Example 5

This example computes the spacing between metal polygons both as edges (MET1\_SPACE) and as error clusters (MET1\_SPACE1). It then uses the edge layer MET1\_SPACE to select the entire edges of the metal wires that have spacing violations anywhere along the wire (MET1\_SIDES). These “side” edges of the wires are annotated with DFM scores which are derived from measurements of both error clusters and edges, the scores are then transferred onto the metal polygons themselves.

```
MET1_SPACE = EXT [MET1] <= 1.5 OPPOSITE //edge data
MET1_SIDES = TOUCH EDGE MET1 MET1_SPACE
MET1_SPACE1 = EXT MET1_SPACE <= 1.5 OPPOSITE //error clusters
MET1_SIDES1 = DFM PROPERTY MET1_SIDES MET1_SPACE1 OVERLAP ABUT ALSO
    MULTI [ S = EC(MET1_SPACE1)*LENGTH(MET1_SIDES) ]
MET1_SCORE = DFM PROPERTY MET1 MET1_SIDES1 OVERLAP ABUT ALSO
    MULTI [ S = PROPERTY( MET1_SIDES1, S ) ]
```

### Example 6

In the following example, DFM Property is used to split a VIA layer into multiple layers as follows: assume that after oversizing VIA polygons by a certain size, some of the oversized polygons merge together. The sample rules fragment the VIA layer in such a way that each new layer can be sized separately without merging adjacent oversized vias. Note that in this example we assume that at most two VIA polygons can merge together after sizing; a more robust implementation can be done using YieldServer where the number of split layers can depend on the layout. Since the arbitrary selection of one VIA polygon by DFM Property is the desired result, the warning is suppressed using the NOMULTI keyword.

```
// Size up the vias. Enclosures from overlapping vias will merge.

ALL_ENC = SIZE VIA BY 1.5 INSIDE OF MET1 STEP 0.15

// Split via layer so that enclosures produced by vias on the same layer
// do not overlap.
// Use DFM PROPERTY without MULTI to arbitrarily select one via from a
// group of vias whose enclosures overlap.

VIA1 = DFM PROPERTY VIA ALL_ENC OVERLAP NOMULTI [- = COUNT(ALL_ENC)] > 0
VIA_1 = VIA NOT VIA1
VIA2 = DFM PROPERTY VIA_1 ALL_ENC OVERLAP NOMULTI [- = COUNT(ALL_ENC)] > 0

// Size up each VIA sublayer.

ENC1 = SIZE VIA1 BY 1.5 INSIDE OF MET1 STEP 0.15
ENC2 = SIZE VIA2 BY 1.5 INSIDE OF MET1 STEP 0.15
```

# DFM Property Merge

Layer operation

**DFM PROPERTY MERGE** *layer* [*layer* ...]  
 '[' *property\_name = expression* ']' [ '['!'] *constraint* ] ...

## Parameters

- ***layer*** [*layer* ...]
- A required argument that specifies one or more input layers. At least one input layer must be a special unmerged layer created with the [DFM Text](#) operation, the UNMERGED keyword of the [DFM Expand Edge](#) operation, the REGION UNMERGED keyword of the [DFM Copy](#) operation, or the [DFM Read](#) operation. The output layer contains the geometries of the first input layer specified. That is, if the first input layer is an unmerged layer, the output layer contains the merged geometries of this layer. If the first input layer is a merged layer, the output layer contains the geometries of this layer.
- '[' *property\_name = expression* ']' [ '['!'] *constraint* ] ...
- An optional argument set that defines the properties to be associated with geometries on *layer*. This parameter set follows an identical format to the corresponding parameter set for [DFM Property](#), and supports the same expressions.

## Description

Used for manipulating special unmerged polygon layers created with either [DFM Text](#), [DFM Expand Edge](#) UNMERGED, [DFM Copy](#) REGION UNMERGED, or [DFM Read](#). This operation is the only operation that can accept unmerged polygon layers as inputs and convert them into regular merged layers. (The [DFM RDB](#) operation can accept and write out unmerged polygon layers, and DFM Copy can copy these layers.) The output layer of DFM Property Merge contains the merged geometries of the first input layer. Properties on the output layer are computed from the unmerged input geometries using the specified property expressions; DFM Property Merge supports the same expressions as the [DFM Property](#) operation.

For the purpose of evaluation of these expressions, each output geometry is considered to be clustered with all input geometries from which it is created. Therefore, conceptually, the following operation:

```
DFM PROPERTY MERGE layer [expressions_referring_to_layer]
```

is equivalent to the following DFM Property operation:

```
DFM PROPERTY merged_layer layer OVERLAP  

  [expressions_referring_to_layer]
```

Note that this equivalence should be considered for illustration purposes only — the DFM Property operation cannot be used to transfer properties from an unmerged layer.

### Examples

#### Example 1

The following DFM Property Merge operation merges polygons on the layer p\_um and constructs new layer, p\_m, which contains three properties. For each polygon on layer p\_m, consider the values of property “X” on the unmerged polygons which are combined into this polygon. The three output properties contain the smallest, the largest, and the total value of these values of the property “X”.

```
p_m = DFM PROPERTY MERGE p_um
[ XMIN = MIN( PROPERTY(p_um, "X") ) ]
[ XMAX = MAX( PROPERTY(p_um, "X") ) ]
[ XTOT = PROPERTY(p_um, "X") ]
```

#### Example 2

In the following example, a two-layer DFM Property Merge operation is used to attach polygon properties that are read from the input OASIS file to the output polygons as DFM properties.

```
LAYER metall1 1
prop1 = DFM READ metall1 PROPERTY NUMBER x
metall1_prop = DFM PROPERTY MERGE metall1 prop1
[ v = VECTOR(PROPERTY(prop1,x)) ]
```

## DFM RDB

Layer operation

**DFM RDB** *layer1 filename [layer2 ...]*  
 [NODAL [ALL | OTHER]] [NOPSEUDO] [NOEMPTY]  
 [OUTPUT] [MAXIMUM ALL | MAXIMUM *value*] [JOINED]  
 [ALL CELLS] [SAME CELL] [TRANSITION] [ABUT ALSO]  
 [RESULT CELLS *cell\_list*]  
 [CHECKNAME *name*]  
 [COMMENT “*comment\_string*”]  
 [ANNOTATE ‘[’ “*name*” = “*value*” ‘’]...]

or

**DFM RDB** *layer1 NULL [layer2 ...]*  
 [NODAL [ALL| OTHER]]  
 [COMMENT “*comment\_string*”]  
 [ANNOTATE ‘[’ “*name*” = “*value*” ‘’]...]

or

**DFM RDB** *layer1 filename CSG*  
 [NOPSEUDO]  
 [COMMENT “*comment\_string*”]  
 [ANNOTATE ‘[’ “*name*” = “*value*” ‘’]...]

or

**DFM RDB** {GDS | OASIS} {*filename* | DEFAULT} *layer1* [*layer1\_number*  
*[layer1\_datatype]*] [*layer2* [*layer2\_number* [*layer2\_datatype*]] ...]  
 [USERMERGED | PSEUDO | USER] [KEEPEMPTY]  
 [MAXIMUM {*maxresults* | ALL}]

Used in Calibre nmDRC applications (non-nodal), and in Calibre YieldEnhancer applications (nodal).

### Summary

Writes original or derived layer data to the specified database.

### Parameters

- *layer1*

- A required argument specifying the name of the layer to be written to the database.
- If the optional keyword NODAL is specified, this first input layer must contain connectivity data.
  - If the TRANSITION keyword is specified, this first input layer plays the role of a via layer and contains the added vias.

- If additional layers are supplied as [*layer2* ...], this first input layer plays the role of the driving layer (refer to [Partitioning by Clusters](#)).

This layer can be an original or derived layer. Note that edge and error layers are only supported for single layer RDBs.

**Table 4-11. Layer Types Supported by DFM RDB**

DFM RDB	polygon	edge	error
Single-Layer DFM RDB Mode	YES	YES	YES
Single-Layer DFM RDB Mode with NODAL keyword	YES	YES	NO
Multi-Layer DFM RDB Mode	YES	NO	NO
Multi-Layer DFM RDB Mode with NODAL keyword	YES	NO	NO
Transition Mode	YES	NO	NO
Transition Mode with NODAL keyword	YES	NO	NO
With GDS specified	YES	NO	NO

- ***filename***

A required argument that specifies the name of the file to which the layer data is to be written. This argument is ignored when layer data is written to a DFM database. When **GDS** or **OASIS** is specified, the ***filename*** is the name of a GDSII or OASIS file, unless **DEFAULT** is specified. If **DEFAULT** is specified, the operation uses the filename specified in a [DFM Defaults](#) statement.

Within the ***filename***, the [Pattern Substitution](#) facility replaces all occurrences of the pattern “%\_t\_” with the name of the top cell for the design.

You can write the results of multiple DFM RDB operations to a single RDB.

- The first time an RDB file is opened, the system run clears the file and writes the header line consisting of TOP-cell name and precision.
- Subsequent opens are in append mode, and append additional RDB rule checks to the file. The header line is not repeated.

To avoid overwriting results, the RDB file to which you write the data should not be the same file as the nmDRC results database specified in a rule file using the DRC Results Database specification statement. Currently, there is no check that a DFM RDB does not coincide with any other nmDRC results database to be created in the run (outside of DFM RDB) and caution is advised. If an RDB file cannot be opened, then a warning is issued and the operation continues as if RDB were not specified.

- **NULL**

A required keyword used in place of ***filename*** when using DFM RDB specifically for writing layer data to a DFM database. If a rule file containing a DFM RDB NULL operation is processed using calibre -drc, the operation is ignored.

When specified, multiple input layers do not need to be of the same layer type. The following keywords cannot be used with NULL:

ABUT ALSO, ALL CELLS, MAXIMUM, MAXIMUM ALL, NOPSEUDO, NOEMPTY, SAME CELL, and TRANSITION.

- *layer2* ...

The names of one or more optional layer to be written to the database, in addition to *layer1*. These layers must be original or derived polygon layers.

- If the NODAL ALL keywords are specified, connectivity is required on all input layers.
- If the TRANSITION keyword is specified, you must specify exactly two additional layers. These layers represent the added metal enclosure for the vias on *layer1*.
- If the TRANSITION keyword is not specified, these layers play the role of the driven layers (refer to [Partitioning by Clusters](#)).

- NODAL [ALL | OTHER]

An optional keyword used to control whether or not connectivity data is stored in the RDB. When specified, the operation attaches NET and NETNAME properties to geometries in the RDB if that information was available on the input layer(s).

When no secondary keywords are specified, the operation attaches connectivity data to geometries from *layer1*. If the NODAL ALL keyword is specified, the operation attaches connectivity data to geometries from all input layers. If the NODAL OTHER option is specified, the operation attaches connectivity data to geometries from all layers except *layer1* (the driving layer).

The optional secondary keywords ALL and OTHER can be used only when operating in the multi-layer mode. Connectivity is required for the layers to which connectivity data will be attached.

**Table 4-12. Summary of NODAL Options**

	Mode	Use with TRANSITION	Connectivity Required on...	Connectivity data saved for...
NODAL	any	yes	first layer only	geometries on first layer
NODAL ALL	multi-layer	no	all layers	geometries on all layers
NODAL OTHER	multi-layer	no	driven layers	all geometries except first layer

- NOPSEUDO

An optional keyword that controls whether or not the RDB is to reflect the pseudo-hierarchy created by Calibre when processing hierarchical data. When specified, pseudocells (cells created by Calibre to facilitate hierarchical processing) are not included and the RDB is

generated as if all results had been flattened up through pseudocells. These pseudocells appear in the RDB by default.

- **NOEMPTY**

An optional keyword used to control whether or not empty cells are written to the RDB. When specified, rule checks representing cells with no results in the cell's immediate hierarchy are omitted from the RDB.

- **OUTPUT**

An optional keyword that when specified, generates a non-empty output layer. This layer has one rectangle that represents the extent of the results written out by the DFM RDB operation. If the DFM RDB operation does not write any results, the result layer is empty. Note that this keyword has no effect in Calibre -dfm mode, since the behavior of the DFM RDB operation in this mode is to save its input layers to the DFM database.

- **[MAXIMUM ALL | MAXIMUM *value*]**

An optional keyword which used to control the maximum number of results (polygons on the input layer) written to the RDB. If not specified, the operation writes all polygons on the input layer to the RDB.

- **MAXIMUM ALL** — This keyword indicates that all results be output, with no limit (except the disk space of your host machine). This is the default.
- **MAXIMUM *value*** — Limits the number of results (polygons) written to the RDB to *value*. Note that the MAXIMUM value applies to the entire RDB, not to each check. When the MAXIMUM value is reached, no more results are written into the RDB and a warning is printed in the transcript, for example:

```
WARNING: Maximum result count of 1000 exceeded in DFM RDB for layer  
MET1::<1>
```

- **JOINED**

An optional keyword that when specified, outputs error clusters with additional lines joining the edges. These added lines convert error clusters into contiguous chains of edges and line segments. This keyword is useful for identifying the edges that comprise a particular error cluster.

- **ALL CELLS**

An optional keyword used to force the results data from all cells to be written out as a single check. The default is to create one rule check for each cell in the layout database.

When you specify ALL CELLS, the operation adds an additional property to the first result in each cell (polygon, edge, or error cluster). This property is named CN and holds three values, cell name, the inverse to-cell transform, and flat placement count for the current cell, which shows how many instances of the cell are in the entire design.

Note that geometry coordinates are always written in chip-level coordinates, regardless of whether ALL CELLS is specified or not.

ALL CELLS cannot be used with TRANSITION mode.

- SAME CELL

An optional keyword used to disable hierarchical promotion when generating a [Multi-Layer Geometric RDB](#). Specifying this keyword limits clusters to the same level of hierarchy, that is, the cluster is assembled in the same cell as the driving polygon, and only the driven polygons in that cell are added to the cluster. Driven polygons that overlap a driving polygon but belong to a different cell are not assembled into any cluster. This keyword can be specified only when you supply multiple layers as input to the operation.

By default, the operation performs a hierarchical promotion when assembling clusters for multi-layer RDBs.

- TRANSITION

An optional keyword used to indicate that the data is to be written using the by cell format used by the DFM Transition operation. When this keyword is specified, you must supply exactly three input layers for the operation.

- ABUT ALSO

An optional keyword used to specify that driven polygons that abut with a driving polygon, that is, share only an edge or a part of an edge, are grouped with the driving polygon. By default, abutting driven polygons are not assembled in a cluster.

- COMMENT “*comment\_string*”

An optional keyword and argument used to add comments to the RDB. When present, the operation writes *comment\_string* as check text in the RDB (see [RDB Formats Overview](#)). An arbitrary number of COMMENT is allowed and each one adds one line of check text to the appropriate rule check(s).

The *comment\_string* must be a single string, so comments containing spaces or special characters must be enclosed in double quotes (“ ”). While required only for comments containing spaces, double quotes are recommended for all comments. [Pattern Substitution](#) is supported.

If this keyword is encrypted in the rule file, the check comment in the RDB is replaced with <ENCRYPTED COMMENT>.

COMMENT is supported for [Single-Layer DFM RDB Mode](#) and [Multi-Layer DFM RDB Mode](#). It cannot be used with TRANSITION.

- RESULT CELLS *cell\_list*

Optional keyword that controls expansion of results into a set of result cells specified by *cell\_list* (where the *cell\_list* is defined by a [Layout Cell List](#) statement). When this keyword is used, results are partially flattened in such a way that only the specified result cells contain results. Results from any cell not listed as a result cell are expanded into the nearest containing result cell. The top cell is always considered to be a result cell even if it is not explicitly specified as such, and so all results that are not contained within one of the specified result cells are reported in the top cell. This keyword implicitly activates the NOPSEUDO option, since pseudo cells cannot be specified as result cells.

Note that Calibre performs extensive hierarchy transformations, and there is no guarantee that any specified result cell is still present after these transformations. The [Layout Preserve Cell List](#) specification statement, which also uses the Layout Cell List facility, can be used to suppress expansion of the result cells. However, enforcing preserved cells can have a significant negative effect on Calibre performance.

- **CHECKNAME *name***

An optional keyword used to control the name of the rule checks under which results are stored in the RDB. The *name* is a user-specified string. It must be a single string, so comments containing spaces or special characters must be enclosed in double quotes (""). [Pattern Substitution](#) is supported.

By default, the CHECKNAME *name* is generated from name of the derived layer created by the rule check or layer statement in which the operation was executed.

**Table 4-13. Rule Check Names for RDBs Generated by DFM RDB**

	<b>Default</b>	<b>ALL CELLS</b>
Default	checkname::<N>_cellNumber	checkname::<N>
Using CHECKNAME	<i>name</i> _cellNumber	<i>name</i>

where N is an integer indicating that this is the N<sup>th</sup> layer generated for the rule check statement.

- **ANNOTATE [ “*name*” = “*value*” ]**

An optional keyword and arguments used to associate meta data with the input layer(s) when writing them to a DFM database. This keyword is processed only when running calibre with the -dfm switch. It is ignored when running with the -drc switch.

Within the arguments for this keyword, the following constraints apply:

- The square brackets [ ] are required.
- The name can be a quoted or unquoted string, and need not be unique.
- The value must be a quoted string.

You can specify this keyword as many times as needed. Each occurrence adds another annotation (*name* and *value* pair) to the layer.

If an input layer specified for this operation already exists in the DFM database, the ANNOTATE keyword is the only keyword evaluated for that layer. In this situation, the annotations are added to the existing layer in the DFM database. Previously existing annotations are unaffected.

- **CSG**

Required keyword used to write files compatible with the LFD CSG flow. These files are used to annotate device properties computed by Calibre LFD in a subsequent Calibre LVS run. When **CSG** is specified, the DFM RDB operation must be given exactly one input layer

(of any type). Only the NOPSEUDO, COMMENT, and ANNOTATE options are compatible with the CSG option, no other options can be specified.

In CSG mode, the input layer must be annotated with one or more DFM properties. Only numeric polygon properties are considered. Other properties, such as vector properties, NETID properties, or BY NET properties are ignored. For each geometry on the input layer, one line is written to the output file for each property. The format of this line is as follows:

```
LAYOUT TEXT value X Y name cell
```

**Note**

Each line is a valid Layout Text specification statement.

where:

- *value* — the value of the property converted to meters (SPICE unit of length). This conversion takes into account the value of Unit Length if it is specified in the rules (the default unit length is 1e-6).
- *X Y* — the coordinates of the center of the extent of the geometry in user units (polygon, edge, or edge cluster, depending on the input layer type). Conversion to the user units takes into account Precision and DRC Magnify Results magnification, if specified). All coordinates are at the chip level.
- *name* — the name of the DFM property.
- *cell* — the name of the topcell containing the geometry. The cell name is omitted if Calibre is running flat.

The lines for all properties of one geometry are written one after the other, then the lines for the next geometry. The following header is written before the first Layout Text line:

```
// count: time
// comment
// IL: layer
```

**Note**

Each line is an SVRF comment.

where:

- *count* — the number of Layout Text lines written by this DFM RDB operation.
- *time* — the date and time of the run.
- *comment* — the comment string specified in the DFM RDB command. One line is written for each specified COMMENT.
- *layer* — the name of the input layer of the DFM RDB operation.

The same file name can be specified in more than one DFM RDB CSG command, the output generated by all operations is accumulated. The file cannot be used with the other syntax forms of DFM RDB.

- **GDS | OASIS**

Required keyword set that enables generation of a GDSII or OASIS database. This keyword set is intended primarily to be used with the [DFM Fill](#) COMPRESS keyword. Note that the values of all optional keywords must match for multiple DFM RDB GDS or DFM RDB OASIS operations with the same *filename*. In Calibre -dfm mode, the input layers are written to the DFM database. Note that DFM properties, layout properties, and text labels are not preserved with this keyword set.

- *layer1\_number [layer1\_datatype]] [layer2 [layer2\_number [layer2\_datatype]] ...*

Optional layer numbers and datatypes that can be specified with one or more input layers. All input layers must be polygon layers when **GDS** or **OASIS** is specified.

- **USERMERGED**

Optional keyword that instructs the operation to suppress the output of pseudocells in the hierarchical database. Geometries in pseudocells are transformed up the hierarchy to the first user cell, then are merged. This keyword applies to Calibre nmDRC-H only and is the default behavior.

- **PSEUDO**

Optional keyword that instructs the operation to include pseudocells in the hierarchical database. This keyword only applies to Calibre nmDRC-H only.

- **USER**

Optional keyword that instructs the operation to suppress the output of pseudocells in the hierarchical database. Geometries in pseudocells are transformed up the hierarchy to the first user cell and are not merged.

- **KEEPEMPTY**

Optional keyword used to control whether or not empty cells are written to the database. When not specified, cells with no results in their immediate hierarchy are omitted from the database.

- **MAXIMUM {*maxresults* | ALL}**

Optional keyword set that allows you to specify the maximum number of nmDRC results output for the specified input layers. Possible choices are:

- *maxresults* — specifies the maximum shape count for the input layers, where *maxresults* is a nonnegative integer. When the maximum is reached, Calibre issues a warning and no further results are added to the database.
- **ALL** — specifies no limit on the result count (subject to the memory capacity of the host platform) for the layers.

## Description

Writes original or derived layer data to the specified database. This can be either a DFM database, an RDB, or a GDSII file. This statement must occur inside a rule check.

- The operation writes the layer data to a DFM database whenever the rule file containing the operation is processed using “calibre -dfm”.
- The operation writes the layer data to an RDB when the DFM RDB operation contains a filename and either of the following is true:
  - The rule file is processed using “calibre -drc”.
  - The DFM RDB operation is used inside the YieldServer NewLayer command, and the keyword -rdbs\_as\_files is also supplied.
- The operation writes the layer data to a GDSII file when the GDS keyword is specified and the rule file is processed using “calibre -drc”.

When a filename is specified, the operation writes layer data to that file, as an RDB (see [Results Databases](#)). The data format is consistent with RDB data generated by other operations, though specific to the requirements of this particular operation. The RDB created by this operation uses the precision and magnification defined through the DRC Precision and DRC Magnification statements. That is, the precision is printed at the top of the RDB and magnification factor is applied to all results, with rounding to the nearest integer as needed.

This operation operates in any of three modes:

- [Single-Layer DFM RDB Mode](#) — Writes a single layer as a single-layer RDB, or as a layer in a DFM database.
- [Multi-Layer DFM RDB Mode](#) — Writes multiple layers as a multi-layer RDB or as multiple layers in a DFM Database.
- [Transition Mode](#) — Writes three layers in DFM Transition format.

You control the mode through the number of layers you supply and the keywords you include. Each mode is described in its own section.

While technically a layer constructor operation, the derived layer created *in memory* by this operation is always empty and does not persist. The recommended way to use this operation is to include it in a rule check, which will also be empty. For example:

```
RDB1 { DFM RDB M1 "met1.db" NOPSEUDO NOEMPTY }
RDB2 { DFM RDB (NET M2 "A?") "met2_net.db" NODAL NOPSEUDO NOEMPTY }
```

### [Single-Layer DFM RDB Mode](#)

When only one layer is specified, the operation creates either a layer in the DFM database or an RDB file containing one rule check for each cell in the layout. Note that the following keywords should not be present in a single-layer DFM RDB operation and will generate a compilation error if used: NODAL ALL, SAME CELL, TRANSITION, ABUT ALSO.

## Multi-Layer DFM RDB Mode

When multiple layers are specified, the operation either saves the layers in the DFM database or writes the layer data from all the layers into the same RDB file. The RDB contains one rule check for every cell in the layout. Within the rule check, cell data is organized by clusters representing the results from several layers, as described in [Multi-Layer Geometric RDB](#).

## Transition Mode

When TRANSITION is specified, the operation writes data in the [Transitions RDB](#) format, which is a format designed for storing layout modifications (geometries to be added to a layout), such as the results from processing with the DFM Transition operation. This mode lets you:

- Generate layers representing layout modifications using the method of choice, then save them in a familiar format for viewing and manipulation using analysis and post-processing tools, such as Calibre RVE.
- Perform post-process on layers generated by the DFM Transition operation and then write the post-processed geometries in the format that would have been used had they been written to the RDB from DFM Transition directly.

There are several restrictions on the TRANSITION mode DFM RDB operation:

- The operation must have exactly three input layers
  - the first input layer plays the role of a via layer and contains the added vias.
  - the two additional input layers contain the added metal enclosure for these vias.
- The NODAL ALL keyword cannot be used (NODAL keyword can be used)

These restrictions are enforced by the rules compiler.

The format of an RDB generated by the DFM RDB TRANSITION operation is identical to the format of an RDB generated by the DFM Transition operation in BY CELL mode with the following exception: the statistical data present in the check text of a DFM Transition RDB are not available in the DFM RDB TRANSITION RDB, zero values are written out instead.

The optional NODAL keyword causes the RDB to be annotated with net numbers and names, in the same format as an RDB produced by a DFM Transition operation. Only the net numbers from the first input layer, which should be the via layer, are used.

Optional SAME CELL and ABUT ALSO keywords are supported in TRANSITION emulation mode, and have the same effect on the clustering algorithm as they do in a regular multi-layer RDB operation (note the current limitation on the hierarchical clustering algorithm, as previously stated).

## Pattern Substitution

Arguments of the COMMENT and CHECKNAME keywords support pattern substitution for the following patterns:

- %\_t\_ — Replaced by the top cell name.

- `%_l_` — Replaced by the input layer name of the DFM RDB operation. If a DFM RDB operation contains multiple input layers, this pattern is replaced by the first (primary) layer.

The patterns are case-insensitive (for example, both `%_t_` and `%_T_` are equally valid patterns).

## Examples

```
// creates an RDB file with check "M12" and check comment "Layer M12 in
// cell ram1"
LAYOUT PRIMARY "ram1"
M12 = MET1 AND MET2
DFM_CHECK {
  DFM RDB M12 "dfmrdb" CHECKNAME "%_l_" COMMENT "LAYER %_l_ in cell %_t_
}
```

## DFM Read

Layer operation

**DFM READ** *layer { PROPERTY { STRING | NUMBER | VECTOR | VSTRING } { property\_name | property\_attribute property\_name } ... }*

### Parameters

- ***layer***  
A required argument that specifies the input layer, which must be an original layer.
- **STRING | NUMBER | VECTOR | VSTRING**  
A required argument set that indicates the type of DFM property that is created from the layout property.
  - **STRING** — specifies that a string-type DFM property is created. The corresponding layout property must be a string (note that all GDS properties are strings, but OASIS properties can be of different types).
  - **NUMBER** — specifies that a numeric-type DFM property is created by converting the layout property into a floating-point number. If the conversion fails, a warning is printed in the transcript and no marker polygon or property value is created. The layout property can be a string, which must contain a valid number, or a numeric layout property (OASIS only).
  - **VECTOR** — specifies that a vector of numeric-type DFM properties is created by converting one or more layout properties with the ***property\_name*** into floating-point numbers. If only one value exists for the property name, a vector of tuple size 1 is created. If a particular polygon does not have any ***property\_name*** values, a marker is created containing an empty vector.
  - **VSTRING** — specifies that a vector of string-type DFM properties is created. The corresponding layout property must be a string (note that all GDS properties are strings, but OASIS properties can be of different types). If only one property value exists for the ***property\_name***, a vector of tuple size 1 is created. If a particular polygon does not have any ***property\_name*** values, a marker is created containing an empty vector.
- ***property\_name | property\_attribute property\_name***  
A required argument set that identifies the layout property by its name or attribute.
  - ***property\_name*** — used if the layout system is OASIS; specifies the name of the OASIS property. A DFM property of the same name is created on the output layer of the DFM Read operation. Note that property names are case-sensitive.
  - ***property\_attribute property\_name*** — used if the layout system is GDS. The ***property\_attribute*** argument is the number that identifies the GDS property, and the ***property\_name*** argument is the name of the DFM property created on the output layer of the DFM Read operation.

## Description

Accesses layout properties in the input database as DFM properties. This operation creates a special unmerged polygon layer containing small marker geometries at the locations of the layout properties on the specified input layer.

Any number of **PROPERTY** clauses may be specified. If the operation contains several property clauses, the output layer will contain marker shapes annotated with several DFM properties. If at least one of the specified properties is not found on an input layer polygon, or the conversion using the **NUMBER** or **VECTOR** keywords is not valid, both the marker for this polygon and all of its properties are not created.

If the operation creates both scalar (NUMBER or STRING) and vector (VECTOR or VSTRING) properties, all scalar properties must have values; if a layout property specified by a name of a NUMBER or STRING property is missing, the marker is not created, and all other properties that would have been attached to this marker are lost.

The DFM Read operation creates a special unmerged polygon layer. (Because there are no limitations on how close two property markers can end up, the resulting marker polygons can overlap.) The [DFM Property Merge](#) operation is the only operation that can use the unmerged layers as inputs and convert them into regular merged layers ([DFM RDB](#) can write out unmerged layers, and [DFM Copy](#) can copy these layers). When merging layers created by the DFM Read operation, you must specify how to treat the overlapping property markers.

One option is to select one of the properties arbitrarily using the **PROPERTY()** or **SPROPERTY()** function. For example:

```
LAYER prop 1
prop_str = DFM READ prop PROPERTY STRING s
prop_num = DFM READ prop PROPERTY NUMBER x

merged_str = DFM PROPERTY MERGE prop_str
[s = SPROPERTY(prop_str,s,1)]

merged_num = DFM PROPERTY MERGE prop_num
[x = PROPERTY(prop_num,x,1)]
```

It is also possible to detect locations where multiple property markers overlap and either suppress them from the output layer or create a separate layer. For example:

```
merged_unique = DFM PROPERTY MERGE prop_str
[s = SPROPERTY(prop_str,s,1)]
[- = COUNT(prop_str)] == 1
conflicts = DFM PROPERTY MERGE prop_str
[- = COUNT(prop_str)] > 1
```

You can also collect all properties on the set of overlapping markers in a vector property:

```
LAYER prop 1
prop_str = DFM READ prop PROPERTY STRING s
prop_num = DFM READ prop PROPERTY NUMBER x

merged_str = DFM PROPERTY MERGE prop_str
[s = VSTRING(SPROPERTY(prop_str,s))]
```

```
merged_num = DFM PROPERTY MERGE prop_num
[x = VECTOR(PROPERTY(prop_num,x))]
```

For numeric properties, it is possible to perform calculations on the property values associated with the overlapping markers, such as computing the sum or average, or selecting minimum or maximum values:

```
merged_num_calc = DFM PROPERTY MERGE prop_num
[sum_x = PROPERTY(prop_num,x)]
[average_x = PROPERTY(prop_num,x)/COUNT(prop_num)]
[min_x = MIN(PROPERTY(prop_num,x))]
[max_x = MAX(PROPERTY(prop_num,x))]
```

For OASIS layouts, property names may conflict with SVRF keywords, such as NET. To avoid conflicts, such property names must be enclosed in quotes ("").

## Examples

### Example 1

```
// read a property from a GDSII file and write out an unmerged layer
LAYER MAP 65 DATATYPE 20 2000

P_NUM_STR = DFM READ 2000
    PROPERTY NUMBER 12 "number_attr_12x"
    PROPERTY STRING 12 "string_attr_12x"

write_unmerged_prop {
    DFM RDB P_NUM_STR 'layout_prop.rdb' ALL CELLS
}
```

### Example 2

```
// read a property from a GDSII file and write out a merged layer
LAYER MAP 65 DATATYPE 20 2000

P_NUM_STR = DFM READ 2000
    PROPERTY NUMBER 12 "number_attr_12x"
    PROPERTY STRING 12 "string_attr_12x"

P_NUM_STR_M = DFM PROPERTY MERGE P_NUM_STR
    [ number_attr_12x = PROPERTY(P_NUM_STR,number_attr_12x) ]
    [ string_attr_12x = SPROPERTY(P_NUM_STR,string_attr_12x,1) ]

write_merged_prop {
    DFM RDB P_NUM_STR_M 'layout_prop_merged.rdb' ALL CELLS
}
```

# DFM Redundant Vias

Layer operation

## **DFM [NON] REDUNDANT VIAS *layer vra\_spec***

**[HIERARCHICAL | FLAT] [QUANTIFY [RGID] [RGFS] [RGLS]]**

Used only in Calibre YieldAnalyzer applications.

### Parameters

- **NON**

An optional keyword that instructs the operation to identify non-redundant polygons instead of redundant polygons. If this argument is not specified, the operation identifies redundant polygons.

- ***layer***

A required argument that specifies the layer on which redundant (or non-redundant) polygons are identified. This layer is typically a via layer, and must be a [Connect](#) layer.

- ***vra\_spec***

A required argument that specifies the name of a via redundancy analysis specification created with the [DFM Spec Via Redundancy](#) statement.

- **HIERARCHICAL | FLAT**

An optional keyword pair that specifies whether the operation runs in hierarchical or flat mode (the default is hierarchical).

In hierarchical mode, every redundant (or non-redundant) polygon is saved in the lowest-level cell that fully contains the hierarchical net to which the polygon belongs. The layer connectivity defined in the *vra\_spec* must be compatible with the global layer connectivity defined by the Connect statements so that correct connectivity information can be found on all layers specified in the *vra\_spec*. Hierarchical mode supports MT and MTFlex processing (nets are processed in parallel).

In flat mode, the operation processes an internally-flattened layout and produces flat output — all redundant (or non-redundant) polygons are saved in the top-level cell. In this mode, connectivity is not required to be present (no restrictions are placed on the connectivity defined in the *vra\_spec*).

- **QUANTIFY [RGID] [RGFS] [RGLS]**

An optional keyword set that enables the output of additional information for redundant groups of polygons (in terms of vias, a redundant via group is the largest set of vias that are connected together when all single vias are removed). The keywords RGID, RGFS, and RGLS enable the creation of properties on the output layer:

- RGID — when specified, a property named rgid (rgid stands for “redundant group ID”) is created on the output layer. This property contains a numeric value that is assigned to all polygons associated with a particular group of redundant polygons.

The property value may change between runs (especially between multi-threaded runs). Each group is assigned a unique rgid value.

- RGFS — when specified, a property named rgfs (rgfs stands for “redundant group full size”) is created on the output layer. This property contains a numeric value that indicates the size of the redundant group to which a particular polygon is associated with, for all layers in the group.
- RGLS — when specified, a property named rglsls (rglsls stands for “redundant group layer size”) is created on the output layer. This property contains a numeric value that indicates the size (number of shapes) of the redundant group to which a particular polygon is associated with, for the input layer only.

By default, all properties are output when none of the keywords RGID, RGFS and RGLS are specified. The QUANTIFY keyword set applies only to the DFM Redundant Vias operation.

### Description

Identifies redundant or non-redundant polygons on a layout and produces a derived polygon layer containing the identified polygons. A redundant polygon is defined as a polygon that may be removed without affecting the connectivity of the input layers.

---

#### Note



This operation outputs only functional polygons. A functional polygon is a shape that interacts with both the lower and upper layers. Non-functional polygons (shapes that interact with zero or one layers) are ignored. This means that DFM Redundant Vias and DFM NON Redundant Vias may not be complementary operations if a design contains non-functional polygons.

---

This operation offers similar functionality to that of [DFM Connectivity Redundant](#); however, it offers advantages in how hierarchy is treated. Particularly, any connection that causes a redundant via but is outside the cell hierarchy where the original via exists is included in the output. This operation must be used in a hierarchical nmDRC or DFM run; flat runs produce empty output.

The CALIBRE\_DFM\_VRA\_MAX\_EDGES\_PER\_TILE environment variable can be used to improve the scalability of this operation in MT/MTFlex mode. This variable specifies the upper bound for the tile size as the average flat edge count per tile, and should be set to a reasonable (positive) value. The default value is 1,000,000.

### Examples

```
CONNECT M1 M2 BY VIA1

DFM SPEC VIA REDUNDANCY m1m2_connectivity
CONNECTIVITY
    M1 VIA1 M2 ;
END CONNECTIVITY
```

```
// find redundant vias on layer VIA1  
m1m2_redundant_vial = DFM REDUNDANT VIAS VIA1 m1m2_connectivity  
  
// find non-redundant vias on layer VIA1  
m1m2_nonredundant_vial = DFM NON REDUNDANT VIAS VIA1 m1m2_connectivity
```

# DFM Select Check

Specification statement

**DFM SELECT CHECK [NODAL] *rule\_check* [*rule\_check* ...]**

## Parameters

- **NODAL**

An optional argument that instructs the Calibre hierarchical engine to preserve node numbers when processing the operations in the specified check(s). If the same check is selected by both DFM Select Check and DFM Select Check NODAL, the check is treated as nodal. All layers produced in rule checks selected with DFM Select Check NODAL must be nodal layers; otherwise, the application reports the following error:

```
Cannot establish MASK connectivity of the output layer in DFM NODAL RuleCheck.
```

For example, the Flatten operation is not node-preserving and produces this error if it appears in a rule check selected with DFM Select Check NODAL.

This behavior applies to both Calibre -drc and -dfm modes. In Calibre -dfm mode, DFM Select Check NODAL causes selected rule check layer output to be saved to the DFM database with node numbers automatically. However, it does not modify the behavior of DFM RDB such that all layers are saved to the DFM database with node numbers preserved (the DFM RDB NODAL option must be used to enable this behavior).

- ***rule\_check***

A required argument specifying the name of a rule check or rule check group. You must specify at least one ***rule\_check***.

## Description

Allows you to specify DFM rule checks to be executed and saved to the DFM database when Calibre is invoked in -dfm mode.

If DFM Select Check is specified, running Calibre in -dfm mode executes checks from the following set:

- Checks specified with DFM Select Check, minus checks specified with [DFM Unselect Check](#).

If DFM Select Check is not specified, running Calibre in -dfm mode executes checks from the following set:

- Checks specified with [DRC Select Check](#) (if DRC Select Check is not specified, all checks are selected), minus checks specified with [DRC Unselect Check](#), minus checks specified with [ERC Select Check](#), minus checks specified with DFM Unselect Check.

If a rule check is specified with DFM Select Check or ERC Select Check, it will not execute when Calibre is run in -drc mode, unless it is also specified with DRC Select Check.

## Examples

In the following examples, assume Calibre is run in -dfm mode on a deck that contains four checks: check1, check2, check3, and check4.

```
// executes check1 and check3
DFM SELECT CHECK check1 check3

// executes check3 and check4
DFM UNSELECT CHECK check1 check2

// executes check2
DRC SELECT CHECK check2

// executes check4
DRC UNSELECT CHECK check2
DFM UNSELECT CHECK check3
ERC SELECT CHECK check1

// executes check1
DFM SELECT CHECK check1
DRC SELECT CHECK check2 check3
DRC UNSELECT CHECK check1
```

## DFM Shift Edge

Layer operation

**DFM SHIFT EDGE** { '['*layer1*' | *layer1* }  
**{ OUTSIDE BY** *outside\_property* | **INSIDE BY** *inside\_property* }  
[EXTEND BY *extend\_property*]  
[OFFSET *offset\_property*]

Used only in Calibre YieldEnhancer applications.

### Parameters

- { '['*layer1*' | *layer1* }

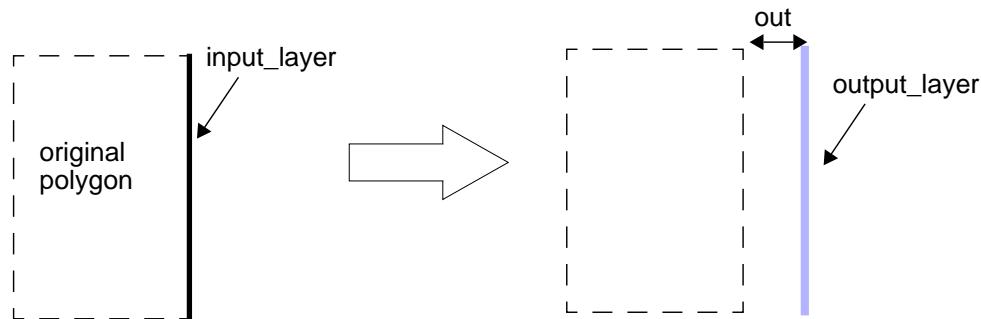
A required input layer that must be a derived edge layer or a derived error layer. If square brackets are used to enclose *layer1*, the output layer is a shifted edge. If square brackets are not used to enclose *layer1*, the output layer is an edge pair that consists of the original edge and the shifted edge.

The names of properties attached to *layer1* can be passed to the operation as the arguments *outside\_property*, *inside\_property*, and *extend\_property*. Alternatively, any of these arguments may be specified as constant values (either directly in the operation or by using a [Variable](#) statement).

If *layer1* is an unmerged edge or error layer that does not contain DFM properties, it is merged before shifting occurs, and the operation outputs a merged layer. If an unmerged *layer1* does contain DFM properties, it is not merged before shifting occurs, and the operation outputs an unmerged layer.

- **OUTSIDE BY** *outside\_property*

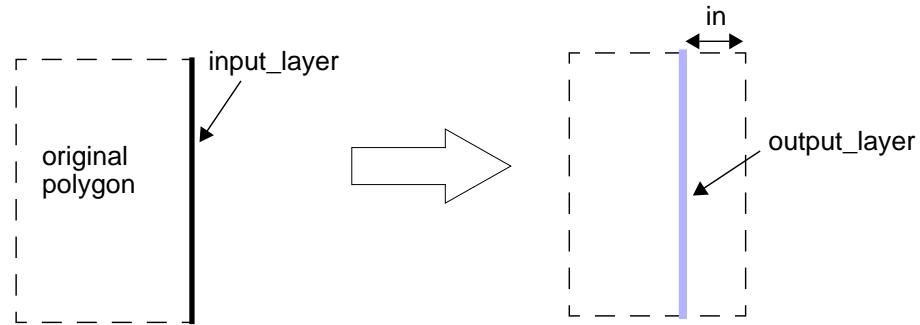
A required keyword and argument that specifies that each edge is shifted in the direction perpendicular to the edge and outside of the original polygon from which it was created. The name of the property that defines the shift distance is specified by *outside\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *outside\_property*.



```
input_layer_p = DFM PROPERTY input_layer [out = LENGTH(input_layer)*0.2]
output_layer = DFM SHIFT EDGE [input_layer_p] OUTSIDE BY "out"
```

- **INSIDE BY *inside\_property***

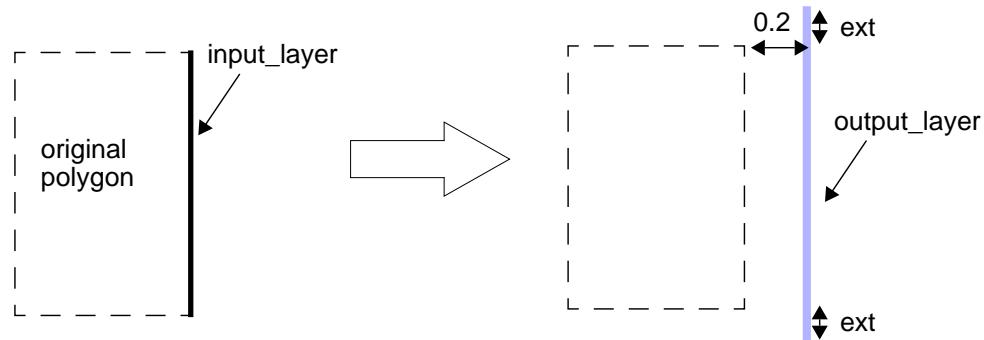
A required keyword that specifies that each edge is shifted in the direction perpendicular to the edge and inside of the original polygon from which it was created. The name of the property that defines the shift distance is specified by *inside\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *inside\_property*.



```
input_layer_p = DFM PROPERTY input_layer [in = LENGTH(input_layer)*0.2]
output_layer = DFM SHIFT EDGE [input_layer_p] INSIDE BY "in"
```

- **EXTEND BY *extend\_property***

An optional keyword that specifies that each edge is extended by *extend\_property* in both directions along the edge before shifting is applied. The name of the property that defines the extension distance is specified by *extend\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *extend\_property*.



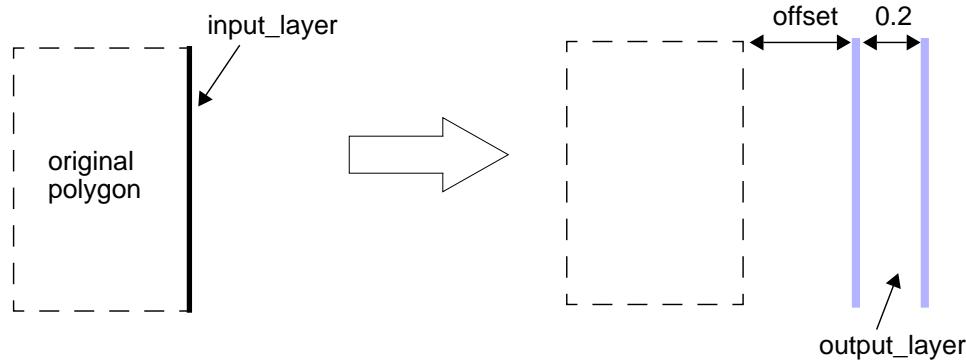
```
input_layer_p = DFM PROPERTY input_layer [out = 0.2] [ext = LENGTH(input_layer)*0.1]
output_layer = DFM SHIFT EDGE [input_layer_p] OUTSIDE BY "out" EXTEND BY "ext"
```

It is possible to extend or shrink an edge without shifting it. To do this, specify 0 for INSIDE BY or OUTSIDE BY along with EXTEND BY. For example:

```
// extends edge layer L by 0.1
L_ext = DFM SHIFT EDGE [L] INSIDE BY 0 EXTEND BY 0.1
```

- **OFFSET *offset\_property***

Optional keyword that causes each output edge pair to be shifted from the original edges. This keyword can be used only if ***layer1*** is not enclosed in square brackets. The name of the property that defines the offset value is specified by *offset\_property*. It must be enclosed in quotes. Alternatively, you may use a constant value for *offset\_property*.



```
input_layer_p = DFM PROPERTY input_layer [out = 0.2] [offset = LENGTH(input_layer)*0.5]
output_layer = DFM SHIFT EDGE input_layer_p OUTSIDE BY "out" OFFSET "offset"
```

If the original edge is shifted in one direction, for example OUTSIDE BY, the edge is first shifted by the OUTSIDE BY value to create an edge pair, then the original edge is shifted by the OFFSET value in the direction of the expansion, creating the gap shown in the figure.

## Description

Shifts edges on the input layer. The result of this operation is either a derived edge layer (if ***layer1*** is enclosed in square brackets) or a derived error layer (if ***layer1*** is not enclosed in square brackets).

At least one of the options **OUTSIDE BY** and **INSIDE BY** must be specified; if either of these is not specified the corresponding shift is not applied (its value is zero). The values of **OUTSIDE BY** and **INSIDE BY** can be positive or negative.

The constant values or the values of the property that define the extension distance, *extend\_property*, can be positive or negative; a negative value causes an edge to shrink (pull in from both ends) instead of extend in both directions along the edge. Similarly, the constant values or the values of the property that define *offset\_property* can be positive or negative.

## Examples

### Example 1

This example shows how to generate an input layer with DFM Property that can be used in a DFM Shift Edge operation.

```
met_edge = m1 LENGTH < 1
met_edge_p = DFM PROPERTY met_edge [ out = LENGTH(met_edge)*.01 ]
met_shift = DFM SHIFT EDGE [met_edge_p] OUTSIDE BY "out"
```

## DFM Size

Layer operation

**DFM SIZE** *layer1* BY *size\_property* {[OVERLAP ONLY] |  
 [{INSIDE OF | OUTSIDE OF} *layer2*] } [STEP *step\_property*] [BEVEL *value*]  
 [TRUNCATE *distance*]

Used only in Calibre YieldEnhancer applications.

### Parameters

- ***layer1***

A required input layer that must be created using a [DFM Property](#) operation. It must have an associated DFM property for *size\_property* or *step\_property*.

- **BY *size\_property***

A required keyword and argument set that specify the value in user units by which to resize each polygon on the input layer. The name of the property that defines the size value is specified by *size\_property*. Alternatively, you may specify a constant value for *size\_property*. Positive values expand each polygon, negative values shrink each polygon, and zero values leave each polygon unchanged.

- **OVERLAP ONLY**

An optional keyword that specifies that the output consists only of regions where the oversized polygons overlap (not the oversized polygons themselves). This is overlap of distinct oversized polygons. You may not specify OVERLAP ONLY with the INSIDE OF or OUTSIDE OF keywords.

- **INSIDE OF *layer2***

Optional keyword followed by an original layer or layer set, or a derived polygon layer, that restricts polygons on *layer1* from expanding outside the boundary of *layer2*. Polygons on *layer1* that are not enclosed by *layer2* are not resized and are excluded from the result layer. This keyword cannot be used with OUTSIDE OF or OVERLAP ONLY.

If a polygon is undersized, it is not constrained by *layer2*, however, it must obey the restrictions on the input polygons. For example, a polygon that is undersized with INSIDE OF must initially interact with a *layer2* polygon, otherwise it is excluded from the result layer.

- **OUTSIDE OF *layer2***

Optional keyword followed by an original layer or layer set, or a derived polygon layer, that restricts polygons on *layer1* from intruding inside the boundary of *layer2*. For each input polygon outside of the *layer2* polygons, the resulting polygon is constrained to remain outside all *layer2* polygons. Input polygons that are enclosed by *layer2* polygons are not resized and are excluded from the result layer. May not be used with INSIDE OF or OVERLAP ONLY.

- **STEP** *step\_property*

An optional keyword that specifies the name of the DFM property associated with *layer1*. Alternatively, you may specify a constant value for *step\_property*. If STEP is specified, resizing for each polygon is done in steps given by the value of *step\_property*, which must be non-negative for both under and oversizing. If this keyword is not specified, or the value of *step\_property* for a particular polygon is zero, the polygon is resized by the entire amount specified by the value of *size\_property* in a single step. If this keyword is used together with one of the bounding keywords, INSIDE OF or OUTSIDE OF, the bounding layer constraint is applied after each resizing step.

- **BEVEL** *value*

An optional keyword and positive integer that allows orthogonal corners (two edges with an interior angle of 90 degrees) on *layer1* to be approximated by segmented arcs on the output layer. The BEVEL *value* is ignored for polygons with a negative size value. BEVEL may not be specified with OVERLAP ONLY.

Refer to [Size](#) for a conceptual description of BEVEL.

- **TRUNCATE** *distance*

An optional keyword and argument set, where *distance* is a dimensionless, non-negative floating-point value that specifies the value by which to multiply the absolute value of *size\_property* in order to get the truncation distance. The default value is  $1/\cos(67.5)$ , which is approximately 2.61.

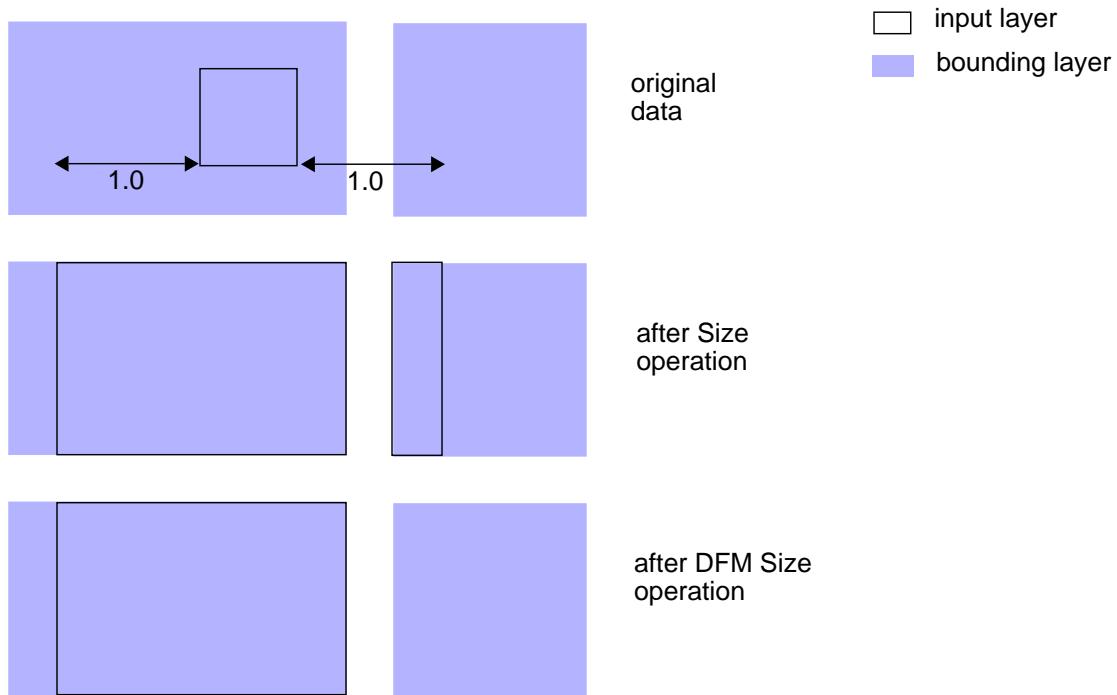
Refer to [Size](#) for a full description of truncation.

## Description

Expands or shrinks polygons on *layer1* by specified amounts. Positive expansion values expand polygon edges outward by that amount. Negative expansion values shrink polygon edges inward by the absolute value of that amount. This operation is similar to [Size](#), however, in addition to taking size and step values from keywords in the operation (either directly in the operation or by using a [Variable](#) statement), DFM Size can also take these values from DFM properties attached to *layer1*, which means they can be different for each polygon. At least one of the parameters *size\_property* and *step\_property* must be specified as a DFM property.

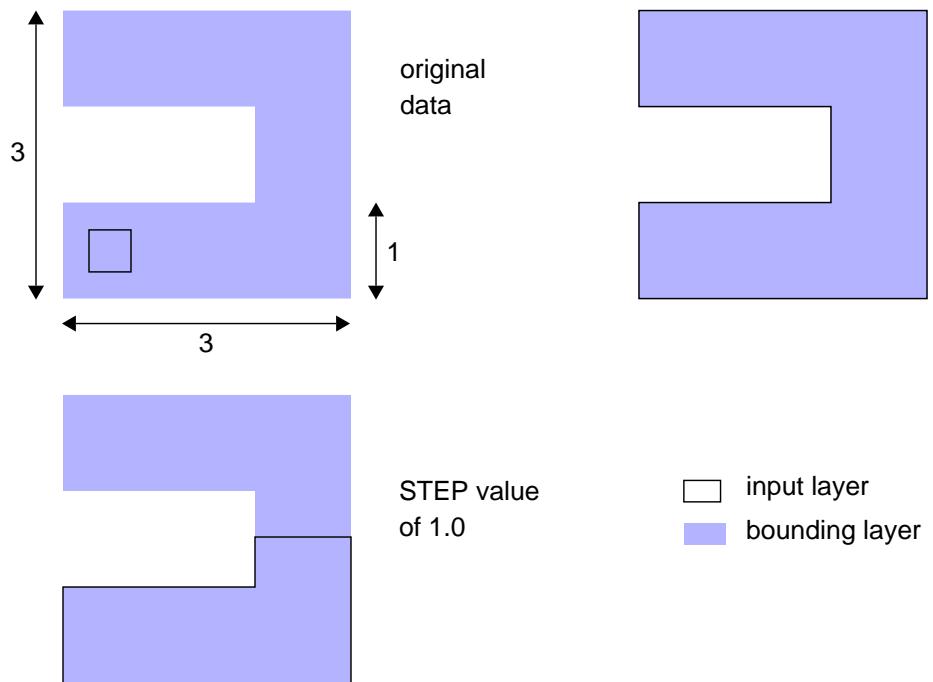
There is a significant difference in behavior between the behavior of [Size](#) and DFM Size with regard to the interaction of bounding layers and the STEP keyword. The [Size](#) operation clips the input polygons to the bounding layer after resizing them by the STEP value, but does not attempt to ensure that oversizing by one STEP value does not extend the input polygon outside of its containing bounding polygon and into the adjacent bounding polygon. Therefore, if the STEP value is not sufficiently small, it is possible for an oversized polygon to extend outside of its containing bounding polygon despite the INSIDE OF option (the oversized result is still contained within the INSIDE OF layer, but not necessarily within the same polygon which enclosed the input polygon). The DFM Size operation oversizes each input polygon inside its corresponding bounding polygon, regardless of the step value.

For example, assume that the input polygon is oversized by 1.0 user units with a step of 1.0 and INSIDE OF the bounding layer. If the oversizing is done using the Size operation, the result consists of two polygons because oversizing the input polygon by one step extends it into the adjacent bounding polygon.



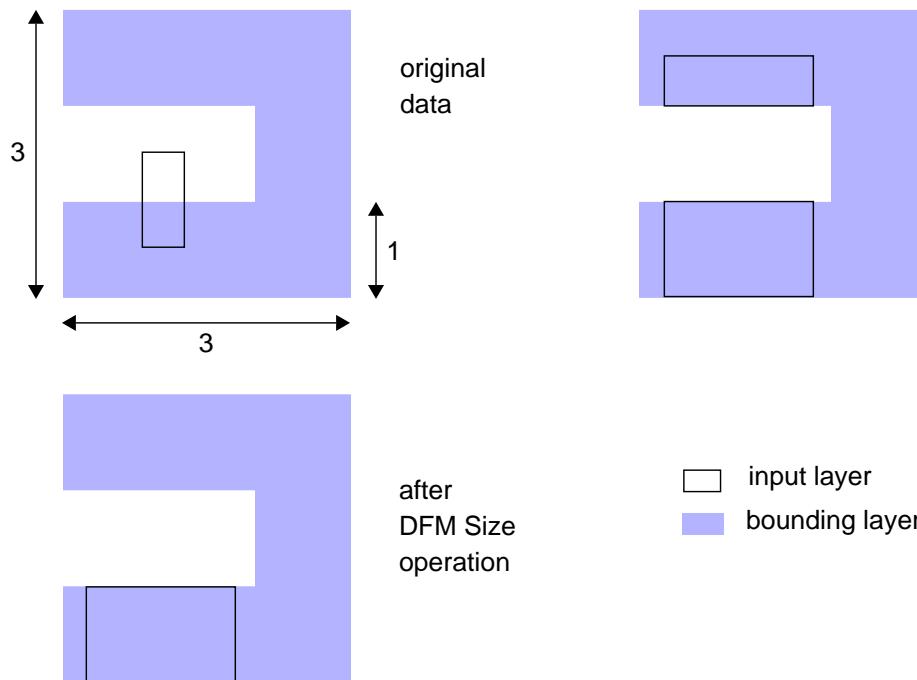
In order to contain the input polygon inside its enclosing bounding polygon, the STEP value must be smaller than the gap between the two bounding polygons. However, if the oversizing is done with the DFM Size operation, then for the same size and step values, the result has only one polygon. Therefore, it is not necessary to use STEP in the DFM Size operation to ensure proper containment of the oversized polygons.

The STEP keyword affects the results of DFM Size in other ways. For example, assume that the input polygon is oversized by a value of 3.0 user units INSIDE OF the bounding layer. If the STEP value is not specified, the oversizing is done in one step, and the result is the portion of the oversized input polygon clipped by the bounding polygon. However, if the same operation is done with the STEP option and the value of *step\_property* is 1.0, then the oversizing is done in three steps and the oversized polygon “follows” the bounding polygon.



Another difference between DFM Size and Size is that for Size, an input polygon is resized before being clipped by the bounding layer. With DFM Size, it is clipped before resizing and again after resizing. This means that if an input polygon is entirely outside of the bounding layer, it won't grow into the bounding area to generate a result. Also, an input polygon that

partly overlaps a bounding polygon can generate different results with DFM Size than with Size. For example:



## Examples

This example demonstrates how to create an input layer with DFM Property and use it in a DFM Size operation.

```
// expand each polygon on layer m1 by its area in one step
m1_prop = DFM PROPERTY m1 [ SIZE_VALUE=AREA(m1) ]
m1_expand = DFM SIZE m1_prop BY SIZE_VALUE
```

# DFM Space

Layer operation

**DFM SPACE** *layer1* [*layer2*] *distance\_constraint*  
{ BY EXT | BY INT | BY ENC | ALL }  
[HORIZONTAL VERTICAL | HORIZONTAL | VERTICAL]  
[COUNT *shielding\_constraint*] [BY LAYER *layer*] [MEASURE ALL]  
[[NOT] CONNECTED] [GRID *step*]

## Parameters

- ***layer1* [*layer2*]**

A required argument set that specifies one or two input layers. The input layers must be polygon or derived edge layers.

- ***distance\_constraint***

A required argument that specifies a constraint on the distance between two edges, which must be satisfied for the edges to be part of the output. The distances are measured only between edges that project either vertically or horizontally onto each other. The distance between edges is measured along the horizontal and vertical directions. Note that the edges themselves do not have to be orthogonal — angled edges on the input layers are allowed. However, all distances are measured vertically or horizontally, even between angled edges.

- **BY EXT | BY INT | BY ENC | ALL**

A required keyword set that specifies the type of measurement to be performed, corresponding to the [External](#), [Internal](#), and [Enclosure](#) operations:

- **BY EXT** — only the external distances between two outside edges are measured.
- **BY INT** — only the internal distances between two inside edges are measured.
- **BY ENC** — only the enclosure distances between an outside and an inside edge are measured.
- **ALL** — edge orientation is ignored and the output contains edge pairs that would have been selected by any of the corresponding External, Internal, or Enclosure operations, including edge pairs that would be selected by Enclosure with the layer arguments reversed.

- **HORIZONTAL VERTICAL | HORIZONTAL | VERTICAL**

An optional keyword set that specifies projection constraints. If HORIZONTAL is specified, the distances between edges are measured horizontally. If VERTICAL is specified, the distances between edges are measured vertically. If both are specified, either explicitly or by default, both distances are measured. Note that the direction is that of the measurement, not that of the edge. For example, for orthogonal edges, the DFM Space HORIZONTAL operation measures only vertical edges that have been selected by any of the corresponding External, Internal, or Enclosure operations.

- COUNT *shielding\_constraint*

An optional keyword and parameter used to control shielding. The *shielding\_constraint* is applied to the number of edges directly between the two edges that are considered for output (that is, the edges satisfy all other constraints). Note that when counting the shielding edges, all edges on all input layers are considered. If, for a given edge pair that satisfies all other measurement constraints, the number of the shielding edges satisfies the *shielding\_constraint*, the edge pair is sent to the output layer as an edge cluster. The default shielding constraint is “==0” (that is, direct visibility between the two edges with no intervening edges in between). The shielding constraint can be an equality (for example, ==1), an inequality (for example, <3), or a range (for example, >=1 <4). Partial shielding is supported — if part of the edge is shielded and part is visible, the visible portion is returned as output. Performance of the DFM Space operation depends strongly on the shielding value — large shielding values will cause performance degradation.

- BY LAYER *layer*

An optional keyword and parameter used to control shielding. When specified, only the edges on the specified *layer* are considered as shielding edges. The *layer* specified must be one of the two input layers. The BY LAYER keyword is valid only for two-layer DFM SPACE operations.

- MEASURE ALL

An optional keyword that instructs the operation to ignore the polygon containment criteria when two input layers are specified. This keyword is similar to that for the External, Internal, and Enclosure operations. By default, two-layer measurements must satisfy the polygon enclosure constraint:

- For **BY EXT**, neither edge may be strictly inside a polygon on the other layer (if the other layer is a derived polygon layer) or coincident with an oppositely-oriented edge of the other layer.
- For **BY INT**, neither edge may be:
  - coincident with an edge of the other layer
  - outside all polygons of the other layer (if it is a derived polygon layer)
  - inside a different polygon of the other layer (if it is a derived polygon layer) than the one whose edge it is being measured against.
- For **BY ENC**, the edge on *layer1* (the enclosed edge) must satisfy the BY INT condition; the edge on *layer2* (the enclosing edge) must not coincide with an edge of the same orientation on the other or be strictly inside a polygon on the other layer (if the other layer is a derived polygon layer).

No polygon containment condition is imposed for **ALL**.

- [NOT] CONNECTED

An optional keyword set used as a connectivity constraint. This keyword set restricts the output to edges that are on the same net (when CONNECTED is specified) or on different nets (when NOT CONNECTED is specified).

- GRID *step*

An optional keyword and parameter set that allows for pitch and grid checking. If GRID step is specified, then vertical or horizontal edge pairs separated by a multiple of *step* are not output.

### Description

Allows for very efficient measurements of edge-to-edge distances for large ( $>0.5\text{ um}$ ) distance constraints. This operation performs measurements and produces results that are similar to the DRC operations External, Internal, and Enclosure. The DFM Space operation supports a much more limited set of options than these operations. However, for the measurements that can be done using both the External, Internal, and Enclosure operations and the DFM Space operation, the performance of the DFM Space operation is much better when the measurement distance constraint is large. The output layer is always a derived error layer (the edge output of the External, Internal and Enclosure operations is not supported).

### Examples

```
// generate edge pairs for gates within the optical radius
VARIABLE optradius 2
gate_c1 = DFM SPACE gate <= optradius BY INT HORIZONTAL
```

# DFM Spec Fill

Specification statement

## DFM SPEC FILL *spec\_name*

```
[ {OPTIMIZER optimizer_name} | {optimization_criteria} ]  
[ RDB [MISSEDONLY] [FILLEDONLY] db_name]  
[ WINDOW {w | x y} [STEP sx sy] [BACKUP] ]  
[ GWINDOW {gw | gx gy} [STEP {gs | gsx gsy}] ] [ WRAP ]  
[ {INSIDE OF EXTENT} |  
  {INSIDE OF x1 y1 x2 y2} |  
  {INSIDE OF LAYER layer_analyze [layer_fill]}) ]  
[ INITIAL layer | INITIAL POINT x y | INITIAL FILLREGION ]  
[ PARTITION ]  
fill_pattern_definition [fill_pattern_definition ...]
```

Syntax for *fill\_pattern\_definition*

```
{ FILLSHAPE { OUTPUT output_name “shape_name” } [ [OUTPUT output_name]  
“shape_name”] ... }  
[ STEP step | {xstep ystep} ] [ OFFSET offset | {xoffset yoffset} ]  
[ SETBACK {setback | xsetback ysetback} ]  
[ AUTORotate [E | D | Z] ]  
[ FILLMIN | FILLMAX ]  
[ REPEAT [number] ]  
[ SHAPESPACE [before_spacing] after_spacing ]  
[ EFFORT {1-10}]  
[ SPACE [EDGE] [INTERIOR] space_rule space_layer] ...  
[ ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN  
[constraint] OPPOSITE EXTENDED]
```

*OR*

```
{RECTFILL{[OUTPUT output_name] x1 y1 x2 y2 }...  
| {POLYFILL {[OUTPUT output_name] x1 y1 x2 y2 [xn yn ...] }...  
| {STRETCHFILL {[OUTPUT output_name] x1 y1 x2 y2 max_x max_y  
stretch_incr }...  
}}}  
[ STEP step | {xstep ystep} ] [ OFFSET offset | {xoffset yoffset} ]  
[ SETBACK {setback | xsetback ysetback} ]  
[ AUTORotate [E | D | Z] ]  
[ FILLMIN | FILLMAX ]  
[ REPEAT [number] ]  
[ SHAPESPACE [before_spacing] after_spacing ]  
[ EFFORT {1-10}]  
[ SPACE [EDGE] [INTERIOR] space_rule space_layer] ...  
[ ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN
```

[*constraint*] OPPOSITE EXTENDED]

*OR*

```
{FILLSTACK [UP | DOWN] [MINSTACK min_stack_size]  
[MAXSTACK max_stack_size]  
{OUTPUT [VIA] output_name “shape_name”} ...}  
[ STEP step | {xstep ystep} ] [ OFFSET offset | {xoffset yoffset} ]  
[ SETBACK {setback | xsetback ysetback} ]  
[ AUTORotate [E | D | Z] ]  
[ FILLMIN | FILLMAX ]  
[ REPEAT [number] ]  
[ SHAPESPACE [before_spacing] after_spacing ]  
[ EFFORT {1-10}]  
[ SPACE [EDGE] [INTERIOR] space_rule space_layer] ...  
[ ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN  
[constraint] OPPOSITE EXTENDED]
```

Used only in Calibre YieldEnhancer applications.

## Summary

Defines a fill specification, which details how to create a specific type of fill. Once defined, you can use the [DFM Fill](#) operation to generate a fill layer based on this specification.

## Arguments

Some order-dependence exists within the arguments for DFM Spec Fill: The *spec\_name* and *optimization\_expression* must appear before all other arguments. The *fill\_pattern\_definitions* must appear after all other arguments.

- *spec\_name*

A required argument assigning a name by which this fill specification can be referenced.

- OPTIMIZER *optimizer\_name*

An optional argument supplying the name of the optimizer to use for fill shapes added based on this fill specification. This optimizer is used for all fill shapes specified through the DFM SPEC FILL SHAPE statement.

If you specify OPTIMIZER *optimizer\_name* you cannot also supply an *optimization\_criteria*.

- *optimization\_criteria*

An optional argument, provided for backward compatibility, that allows you to supply an inline definition of how fill shapes should be added based on this fill specification. This is equivalent to defining an optimizer using [DFM Spec Fill Optimizer](#), then referencing it by name. The full syntax for defining an optimizer inline this way is:

['*optimization\_expression*'] [*optimization\_limit*]  
 [GRADIENT *gradient\_constraint* [RELATIVE | ABSOLUTE] [CORNER]]  
 [MAGNITUDE *magnitude\_constraint* [RELATIVE | ABSOLUTE]]

For a complete description of these keywords and arguments, refer to the syntax description for [DFM Spec Fill Optimizer](#).

If you supply an *optimization\_criteria* you cannot also specify OPTIMIZER *optimizer\_name*. Use of OPTIMIZER *optimizer\_name* is recommended going forward.

- RDB [MISSEDONLY] [FILLEDONLY] *db\_name*

An optional keyword, allowed only if an OPTIMIZER or *optimization\_criteria* is specified, that instructs the tool to create a results database file with the specified pathname and write results to that file. Within the *db\_name*, the [Pattern Substitution](#) facility replaces all occurrences of the pattern "%\_t\_" with the name of the top cell for the design.

When specified, this keyword must appear before any *fill\_pattern\_definitions*.

When used with *db\_name* alone, the operation writes detailed statistics for each data capture window to the results database. Statistics include:

- original design densities and gradients
- achieved densities and gradients
- detailed information of gradient dependencies
- potential number of fill shapes of each FILLSHAPE type that could be generated for each window
- number of shapes actually generated and added to the output layer

The secondary keywords, MISSEDONLY and FILLEDONLY, define the windows for which statistics are written to the RDB:

- By default, the RDB contains statistics for each capture window in the design.
- When MISSEDONLY is specified, the operation reports only the statistics for those data capture windows that could not meet density or gradient constraints.
- When FILLEDONLY is specified, the operation reports only the statistics for those data capture windows to which fill was added.
- When MISSEDONLY and FILLEDONLY are specified, the operation reports the statistics for those data capture windows that could not meet density or gradient constraints and those to which fill was added. Statistics for those capture regions that satisfied the constraints without needing any additional fill are not reported.
- WINDOW {*w* | *x* *y*} [ STEP *sx* *sy*] [BACKUP]

An optional keyword set used to specify the size of the data capture window within which the *optimization\_expression* is to be calculated. The operation partitions the design into rectangular areas of the specified size. If WINDOW is not specified, the data capture

window defaults to the analyze extent. The actual analyze extent is dependent upon which INSIDE OF option is specified, if any.

You can specify the window size either of two ways:

- WINDOW  $x\ y$  — the window is a rectangle with  $x$  user units long and  $y$  user units high.
- WINDOW  $w$  — the window is a square with sides  $w$  user units long.

By default, the tool truncates the WINDOW to the boundary size if the WINDOW is larger than the boundary in the x-direction or y-direction.

The secondary keyword STEP and its arguments define an offset for the analysis window. When STEP is specified as a secondary keyword for WINDOW, the ***optimization\_expression*** is evaluated first with analysis window grid of specified size at offset (0,0). The grid is then shifted and optimization is repeated at the following offsets:

(0,0),(sx,0), (2\*sx,0), ... (0, sy), (sx, sy), (2\*sx, sy), ... ( $Nx*sx,Ny*sy$ )

Where:

$$Nx = (wx/sx) - 1$$

$$Ny = (wy/sy) - 1$$

The BACKUP keyword, which is similar to the BACKUP keyword for the **Density** operation, modifies the capture windows on the right and top edges of the optimization area. When BACKUP is specified, these windows are shifted left or down until they align with the right and top edges of the design. Partial windows on the edges of the design are thus eliminated.

- GWINDOW  $gwx\ gwy$  [STEP { $gs$  |  $gsx\ gsy$ }]

An optional keyword used to define a separate grid to be used for optimizing to meet the gradient constraint. If GWINDOW is specified, then gradient optimization will be performed at the grid specified by this argument instead of WINDOW.

The following restrictions are imposed on arguments of WINDOW/GWINDOW options:

- $wx \geq sx, gwx \geq gsx$
- $wy \geq sy, gwy \geq gsy$
- $wx,sx,gwx,gsx$  must be a multiple of the smallest of  $sx$  or  $gsx$ .
- $wy,sy,gwy,gsy$  must be a multiple of the smallest of  $sy$  or  $gsy$ .

**Note**

**Current limitation of WINDOW STEP and GWINDOW:** When these options are used to define step and gradient window sizes that are different from WINDOW size, the optimization\_expression is assumed to be of a straightforward representation of density, such as [ AREA(meta)/AREA() ]. This restriction comes from the need to convert target density values from large WINDOW/GWINDOW to atomic subwindows. This is done by adjusting the value proportionally to the area of the window.

- WRAP

An optional keyword, allowed only when an OPTIMIZER is specified, used to control how the gradient is calculated at the edges of the design.

When you specify WRAP, the gradient calculation for windows at the edge of the design calculates gradient to the window on other side of the design (thus assuming that design will be repeated on the die).

- INSIDE OF EXTENT | INSIDE OF  $x1\ y1\ x2\ y2$  / INSIDE OF LAYER  $layer\_analyze$  [ $layer\_fill$ ]

An optional keyword set that applies constraints to the generated geometry. Only one keyword from this set may be specified. By default (that is, if no keywords from this set are specified), the fill area is the extent of the design database.

- INSIDE OF EXTENT

An optional keyword that specifies that the boundary within which geometries are generated is the extent of all input layers combined. By default, the boundary is the extent of the design database.

Cannot be used with INSIDE OF  $x1\ y1\ x2\ y2$  or INSIDE OF LAYER  $layerA$  [ $layerB$ ].

- INSIDE OF  $x1\ y1\ x2\ y2$

An optional keyword that specifies the lower-left and upper-right corners of a rectangle within which geometries are generated. Coordinates must be specified orthogonal to the database axes, in user units. If a coordinate is negative, it must be enclosed in parentheses ( ). Cannot be used with INSIDE OF EXTENT or INSIDE OF LAYER  $layerA$  [ $layerB$ ].

- INSIDE OF LAYER  $layer\_analyze$  [ $layer\_fill$ ]

An optional keyword that constrains the operation to the areas defined by the specified layer(s). These layers must be original or derived polygon layers.

You can specify either one or two arguments. When one argument is specified, the same layer is used to constrain both the density calculations and fill generation.

When two arguments are specified:

- $layer\_analyze$  defines the boundary areas to be used in density analysis.

- *layer\_fill* defines the boundary areas to be used in fill generation.

It is strongly advised that *layer\_fill* be completely enclosed inside of *layer\_analyze*. Otherwise, the behavior of the command is undefined.

INSIDE OF LAYER cannot be used with INSIDE OF EXTENT or INSIDE OF *x1 y1 x2 y2*.

INSIDE OF LAYER impacts density calculations and fill generation differently.

- For density calculations, INSIDE OF LAYER constrains the analysis to the area within the extent of the layer geometries.

### **Caution**



The INSIDE OF LAYER option for DFM Spec Fill behaves differently from the INSIDE OF LAYER option for the Density operation. For Density, INSIDE OF LAYER constrains the analysis to the extents of each geometry on the layer. For DFM Spec Fill, INSIDE OF LAYER constrains the analysis to the extent of the layer.

- For fill generation, INSIDE OF LAYER constrains the analysis to the areas within the actual polygons on the named layer.

- INITIAL *layer* | INITIAL POINT *x y* | INITIAL FILLREGION

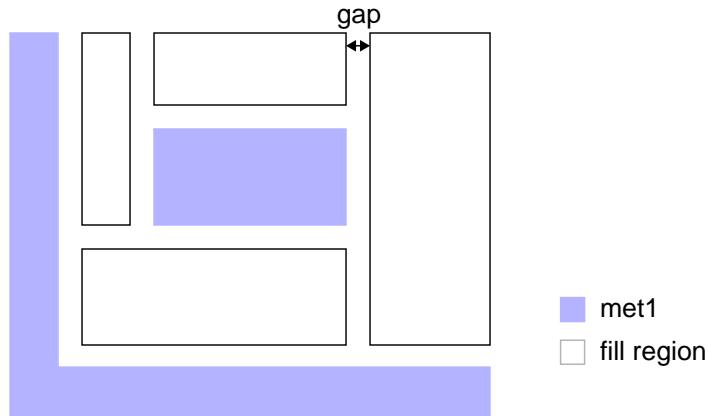
An optional keyword set that enables control of the fillshape placement grid. The grid is aligned differently according to which keyword is specified:

- INITIAL *layer* — uses the lower-left corner of the extent of *layer* as the grid alignment point. This option applies to all fill regions for the operation.
- INITIAL POINT *x y* — uses the specified *x* and *y* coordinates as the grid alignment point. This option applies to all fill regions for the operation.
- INITIAL FILLREGION — uses the lower-left corner of the extent of each fill region as the grid alignment point.

The grid alignment point defines a point that is aligned with the fillshape placement grid. It does not define where the fillshapes are placed initially (that is, it does not define the lower-left corner of the fill region). This keyword set does not affect the spacing of the points on the grid, and is not valid with EFFORT or REPEAT.

- PARTITION

An optional keyword that enables the partitioning of fill into rectangular and triangular regions. A gap is created between adjacent regions to prevent shape spacing violations. The minimum size of each gap is the larger of the *step* and *setback* values. [Figure 4-60](#) shows an example.

**Figure 4-60. DFM Spec Fill PARTITION Example**

- ***fill\_pattern\_definition***

A description of the fill pattern to use when generating fill for the design. You must specify at least one ***fill\_pattern\_definition*** and it must conform to the syntax described in the following section.

The ***fill\_pattern\_definitions*** must be the final section of the DFM Fill specification. There is no upward limit on the number of ***fill\_pattern\_definitions*** you can specify. Fill geometry is generated one ***fill\_pattern\_definition*** at a time, in the order supplied.

#### Syntax for ***fill\_pattern\_definition***

You must begin each fill pattern definition with one of: **FILLSHAPE**, **FILLSTACK**, **POLYFILL**, **RECTFILL**, or **STRETCHFILL**.

```
{ FILLSHAPE { OUTPUT output_name "shape_name" } [ [OUTPUT output_name]
  "shape_name" ] ... ]
[ STEP step | {xstep ystep} ] [ OFFSET offset | {xoffset yoffset} ]
[ SETBACK {setback | xsetback ysetback} ]
[ AUTORotate [E | D | Z] ]
[ FILLMIN | FILMAX ]
[ REPEAT [number] ]
[ SHAPESPACE [before_spacing] after_spacing ]
[ EFFORT {1-10}]
[ SPACE [EDGE] [INTERIOR] space_rule space_layer] ...
[ ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN
  [constraint] OPPOSITE EXTENDED]
```

**OR**

```
{RECTFILL{[OUTPUT output_name] x1 y1 x2 y2 }...
| {POLYFILL {[OUTPUT output_name] x1 y1 x2 y2 [xn yn ...] }...
| {STRETCHFILL {[OUTPUT output_name] x1 y1 x2 y2 max_x max_y
  stretch_incr }...
}}}
```

```
[ STEP step | {xstep ystep} ] [ OFFSET offset | {xoffset yoffset} ]
[ SETBACK {setback | xsetback ysetback} ]
[ AUTORotate [E | D | Z] ]
[ FILLMIN | FILLMAX ]
[ REPEAT [number] ]
[ SHAPESPACE [before_spacing] after_spacing ]
[ EFFORT {1-10} ]
[ SPACE [EDGE] [INTERIOR] space_rule space_layer] ...
[ ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN
[constraint] OPPOSITE EXTENDED]
```

*OR*

```
{FILLSTACK [UP | DOWN] [MINSTACK min_stack_size]
[MAXSTACK max_stack_size]
{OUTPUT [VIA] output_name "shape_name"} ...}
[ STEP step | {xstep ystep} ] [ OFFSET offset | {xoffset yoffset} ]
[ SETBACK {setback | xsetback ysetback} ]
[ AUTORotate [E | D | Z] ]
[ FILLMIN | FILLMAX ]
[ REPEAT [number] ]
[ SHAPESPACE [before_spacing] after_spacing ]
[ EFFORT {1-10} ]
[ SPACE [EDGE] [INTERIOR] space_rule space_layer] ...
[ ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN
[constraint] OPPOSITE EXTENDED]
```

- **FILLSHAPE** {{OUTPUT *output\_name* “*shape\_name*” } ...}

A keyword used to introduce the fill shapes that make up the fill pattern. This keyword is used when the individual layer shapes have been defined using the [DFM Spec Fill Shape](#) statement. Each *shape\_name* specified must be enclosed in quotes.

- For 2-D fill shapes, this keyword is generally followed by a single fill definition (OUTPUT... *shape\_name*). When multiple fill definitions are supplied, the *output\_names* are usually all the same.
- For multi-layer fill shapes, this keyword is followed by two or more fill definitions (OUTPUT... *shape\_name*) and the *output\_names* are all unique.

---

**Note**



A *shape\_name* is associated with the OUTPUT that immediately precedes it. If no such OUTPUT is present, the fill created using this shape will only be returned as output to a DFM FILL command that does not list any outputs. Refer to [DFM Fill](#) for more details.

---

- **POLYFILL** {*x1 y1 x2 y2 [xn yn] ...*}...

A keyword used to introduce the fill shapes that make up the fill pattern that is comprised of one or more polygons when the individual shapes have not been defined using the [DFM Spec Fill Shape](#) statement. The polygons are defined by one or more lists of vertices, which directly follow the keyword.

The operation uses each pair of coordinates as a vertex of a closed polygon. To define multiple polygons for POLYFILL, you must define each polygon in its entirety before defining subsequent polygons. You indicate the completion of one polygon by repeating the final vertex. The vertex after a repeated vertex is treated as the beginning of a new polygon.

- The last vertex in a polygon may or may not match the first one. If it does not, the polygon will be automatically closed; the last vertex will be connected to the first one.
- Edges defined by the vertices must not intersect.
- Polygons with holes in them can be defined by using 0-wide cuts (edge-in is same as edge-out, but in the opposite direction). Refer to [Figure 4-64](#) for an example of this.
- When creating a polygon with holes in it, vertices of the polygon exterior should be enumerated in counter-clockwise direction, whereas inside holes should be enumerated in clockwise direction. This prevents cut lines from intersecting.

You must specify at least one set of vertices. Multiple sets of vertices are allowed. When multiple sets of vertices are supplied, each may be preceded by [OUTPUT output\\_name](#).

- **RECTFILL** {*x1 y1 x2 y2*}

A keyword used to introduce the fill shapes that make up the fill pattern that is comprised of a rectangle when the individual shapes have not been defined using the [DFM Spec Fill Shape](#) statement. The rectangles are defined by a pair of vertices, which directly follow the keyword. The operation uses each pair of vertices as diagonally opposed vertices of a single rectangular fillshape.

- **STRETCHFILL** {*x1 y1 x2 y2 max\_x max\_y stretch\_incr*}...

A keyword used to introduce the fill shapes that make up the fill pattern that is comprised of rectangles that are all variants of a user-defined minimum fill rectangle and the individual minimum fill rectangles have not been defined using the [DFM Spec Fill Shape](#) statement. The stretchable rectangles are defined by one or more lists of seven numeric arguments vertices, which directly follow the keyword.

Each list of seven arguments are interpreted as follows:

- *x1 y1 x2 y2*, defining the minimum fill rectangle,
- *max\_x max\_y*, defining the direction the fill shape is allowed to stretch, and how far it can be stretched in that direction. Only one of values can be non-zero. That non-zero value defines the maximum after-stretch length, in microns, of the fill shape in the associated direction.

For example:

- 0 0.6 — stretch fill shapes in the y direction, with maximum fill shape height of 0.6 microns.
- 0.6 0 — stretch fill shapes in the x direction, with maximum fill shape width of 0.6 microns.

These arguments can be used with FILLSHAPE if *stretch\_incr* is also specified.

- *stretch\_incr*, defining an increment value to be added to a fill shape to stretch it. This argument can be used with FILLSHAPE if *max\_x max\_y* are also specified.
- **OUTPUT *output\_name***  
An keyword and argument that is required when using FILLSHAPE and FILLSTACK and optional when using RECTFILL, POLYFILL, or STRETCHFILL. This keyword is used to control the output of the associated fill.
  - When used with RECTFILL, POLYFILL, or STRETCHFILL, this keyword and argument pair must be placed directly preceding the list of vertices for the shapes used to generate this set of fill. All OUTPUTs introduced by the same RECTFILL, POLYFILL, or STRETCH keyword are assumed to be a single cell, the extent of which is used when determining placement of said geometry.
  - With FILLSHAPE, this keyword and argument pair is required and must precede each *shape\_name*. All OUTPUTS introduced by the same FILLSHAPE keyword are assumed to be a single cell, the extent of which is used when determining placement of said geometry.

The *output\_name* name can be referenced in the [DFM Fill](#) operation to define the exact output to write to a particular results layer. When multiple DFM Fill operations all reference the same DFM Spec Fill, but each specify a different *output\_name*, the operations are processed concurrently, and different types of fill are directed to different output layers.

You can write multiple OUTPUTs to a single layer by including multiple *output\_names* as arguments to a single DFM Fill operation. In this case, geometries from each of the outputs are merged together.

- **FILLSTACK**

A keyword that allows definition of a set of multi-layer fill shapes. This keyword iterates over all allowed combinations of the specified layers and generates a fill pattern for each combination. Refer to [Example 6](#).

- **UP | DOWN**

An optional keyword set that controls the order in which OUTPUT combinations are generated. The default is UP.

- **MINSTACK *min\_stack\_size***

An optional keyword and argument set that specifies the minimum number of metal layers in any one combination. The default is 1.

- **MAXSTACK *max\_stack\_size***

An optional keyword and argument set that specifies the maximum number of metal layers in any one combination. The default is the number of metal layers specified with OUTPUT keywords.

- **OUTPUT [VIA] *output\_name* “*shape\_name*”**

A required argument set similar to the set specified with **FILLSHAPE**. The optional VIA keyword specifies that the output is for a via layer (when specified, the layer is not used in combination generation, but instead used to output its associated geometry only if adjacent layers in the **fill stack** are written out). The *shape\_name* must be enclosed in quotes. Note that for **FILLSTACK**, the order of these argument sets determines how the combinations are generated.

- **STEP {*step* | *xstep* *ystep*}**

An optional keyword defining the distance between adjacent shapes when repeating this fill shape to fill an area, as illustrated in the “[Defining a Simple Array Fill Pattern](#)” section in the *Calibre YieldAnalyzer and YieldEnhancer Reference Manual*. STEP accepts either one argument, indicating that the STEP is the same for both the X and Y directions, or two arguments defining unique values for the STEP in the different directions. When not specified, the default step of 1 dbu is used. Using the default value of step is *not recommended*.

STEP is interpreted to be in the positive x and y directions.

- **OFFSET {*offset* | *xoffset* *yoffset*}**

An optional keyword defining the offsets between each shape and adjacent shapes, as illustrated in the “[Defining a Staggered Array Fill Pattern](#)” section in the *Calibre YieldAnalyzer and YieldEnhancer Reference Manual*. OFFSET accepts either one argument, indicating that the OFFSET is the same for both the X and Y directions, or two arguments defining unique values for the OFFSET in the different directions. When not specified, OFFSET is 0.

OFFSET is interpreted to be in the positive x direction and negative y direction.

When OFFSET is specified, you must consider the STEP and the OFFSET together to fully understand where the next shape will be added. Each fill shape is treated as if it were a cell, with a bounding box being the extent of all the polygons in the fillshape. The location of adjacent fillshape placements are defined by STEP and OFFSET plus the current position of the bounding box for the fillshape and the size of that bounding box. The operation calculates the location of the next fillshape as follows:

Assume (Xcurr, Ycurr) is the location of the current fillshape.

Coordinates of the next shape in the row (shape to the right):

```
Xnext = Xcurr + xstep + sizex
Ynext = Ycurr - yoffset
```

Coordinates of the first shape in the next row (shape going up):

```
Xnext = Xcurr + xoffset  
Ynext = Ycurr + ystep + sizey
```

---

**Caution**

The yoffset for DFM Spec Fill functions differently than for the SVRF [Rectangles](#) operation. If you are making the transition from a legacy SVRF-based fill solution to [DFM Fill](#), be sure to adjust the offset values accordingly.

---

It is possible for multiple STEP values to be defined: Some in DFM Spec Fill Shape statements and some in DFM Spec Fill statements. In this case the more restrictive option wins:

For example, assume the following statements:

```
DFM SPEC FILL SHAPE ONE  
RECTFILL 0 0 1 1 STEP .12  
  
DFM SPEC FILL "OUT"  
FILLSHAPE  
OUTPUT "OUT1" "ONE" STEP .06
```

The geometries in OUT1 will have shape step value of 0.12.

- SETBACK {setback | xsetback ysetback}

---

**Caution**

This option can have a significant impact on runtime and may produce DRC violations in your design. The use of this option is strongly discouraged except under special circumstances.

---

An optional keyword that defines the minimum spacing between fill shapes in adjacent (typically corner-to-corner) fill regions. In certain cases, fill shapes are spaced closer together than the STEP value across these regions, which can cause DRC violations. By default, the post-filtering algorithm removes these violations. The SETBACK keyword allows you to control the behavior of the post-filtering algorithm.

SETBACK accepts either one argument, indicating that the value is the same for both the x and y directions, or two arguments, which allows you to specify unique values for the x and y directions. When SETBACK is not specified, the STEP values are instead used for the spacing checks in adjacent fill regions. Values specified with this keyword must be non-negative. A zero value effectively disables the post-filtering algorithm. If SETBACK is used with either STRETCHFILL or AUTORotate, only one value is allowed for both the x and y directions.

- AUTORotate [E | D | Z]

An optional argument that directs the fill algorithm to attempt to orient fill pattern in either the X or Y direction, picking the direction that satisfies a best fill criteria. The E, D, and Z options operate as follows:

- E — fillshapes are rotated to align the fillshape's major axis to the major axis of each region to be filled. This is the default behavior.
- D — fillshapes are inserted at the rotation that provides the larger count of fillshapes.
- Z — fillshapes are rotated and inserted only into fill regions that received no fill in the default (without AUTORotate) orientation.

AUTORotate can be used with STRETCHFILL, FILLSTACK, or RECTFILL, or with a FILLSHAPE defined using RECTFILL or STRETCHFILL. That is, it cannot be used with either multi-polygonal fill shapes or non-rectangular fill shapes.

- **FILLMIN | FILLMAX**

Optional keywords related to multi-layer fill patterns used to control how the cell created by combining the individual shapes that make up the multi-layer pattern is added with respect to the optimizers.

- FILLMIN (default) — Optimization is considered complete when *at least one* OPTIMIZER referenced by the multi-layer fill shape meets its density/gradient goals. Optimization is performed simultaneously for all layers of the fill shape, which is accomplished by treating all outputs as a single multi-layer cell that is placed as a single unit.
- FILLMAX — Optimization is considered complete when *all* OPTIMIZERS referenced by the multi-layer fill shape meet their density goals. Optimization is performed simultaneously for all OUTPUTs of the fill shape, which is accomplished by treating all outputs as a single multi-layer cell that is placed as a single unit.

- **REPEAT [number]**

Optional keyword that instructs the tool to make multiple passes at filling the specific fill shape until no more shapes are placed. The optional *number* parameter specifies the maximum number of times a fill shape can be repeated. The *number* value must be an integer between 1 and 10, and defaults to 10 if not specified. This parameter allows for a better trade-off between fill density and run-time. In most cases, lower *number* values have lesser impact on run-time while higher values provide for a denser fill. It is recommended that you use the lowest possible *number* value that achieves your desired fill density.

The REPEAT keyword will guarantee at least one fill shape in an appropriately sized empty area (density constraints permitting). Note that there may be a performance penalty when using this option.

- **SHAPESPACE [before\_spacing] after\_spacing**

An optional keyword and parameter set that defines the minimum spacing between the fill geometry generated for this fill pattern and geometry generated by FILLSHAPEs already added to the fill area (*before\_spacing*), and/or all subsequent FILLSHAPEs (*after\_spacing*). The effect of SHAPESPACE is to generate a keep-out area for other FILLSHAPEs. When not specified, SHAPESPACE defaults to the STEP value.

When two SHAPESPACE values affect relation of two fill shapes, the larger of the two space values is used. For example:

```
DFM SPEC FILL m1_fill
...
FILLSHAPE fs_1 ... SHAPESPACE      0.5
FILLSHAPE fs_2 ... SHAPESPACE      0.4
FILLSHAPE fs_3 ... SHAPESPACE 0.6 0.7
```

will result in the following spacing between fill shapes:

```
fs_1 to fs_2 spacing = .5
fs_1 to fs_3 spacing = .6 (larger of .6 and .5)
fs_2 to fs_3 spacing = .6 (larger of .6 and .4)
```

Even though the second SHAPESPACE value for fs\_3, 0.7, is required in order to specify the “before” spacing of 0.6, it is not used.

### **Caution**



When SHAPESPACE is specified with REPEAT, the *after\_spacing* value applies to both subsequent fill shapes and to all subsequent repetitions of the same fill shape.

- **EFFORT** *effort*

An optional keyword and value defining how much computing should be used to optimize fill placement. This keyword lets you make the trade-off between optimal fill placement and run-time.

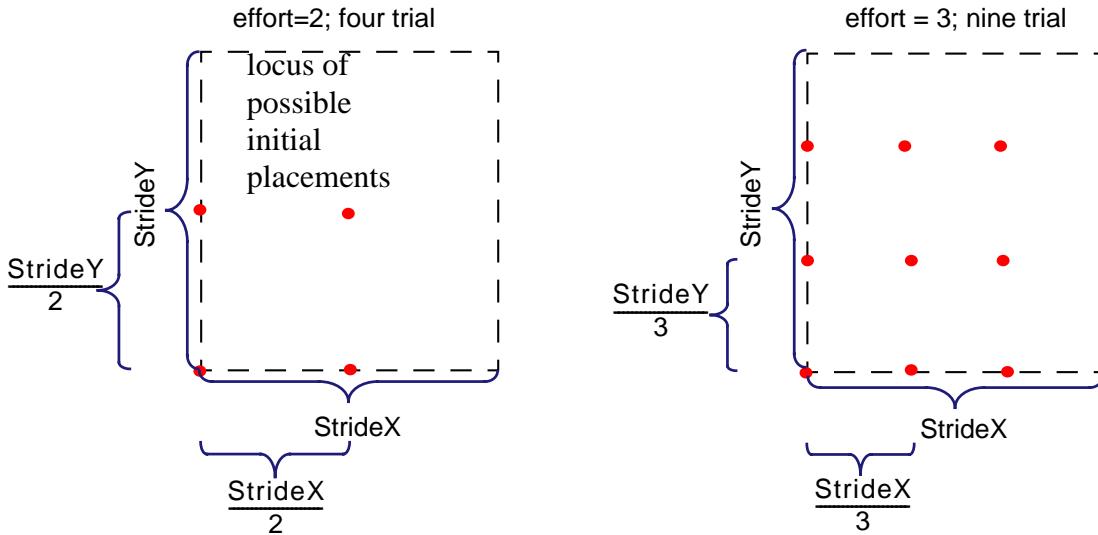
Higher values of EFFORT generate multiple trial placements for the associated fillshape, allowing the operation to select the best fitting pattern. Here, “best fitting” is defined as the one that adds the greatest number fill patterns to a fillable area, while keeping the fill centered in the region. Increasing the EFFORT value instructs the operation to try a larger number of initial fill positions and therefore produce better fitting fill. However, increasing EFFORT will result in a longer run time, with run times for filling a specific area increasing exponentially as EFFORT increases.

EFFORT must be specified as an integer value between 1 and 10. By default, EFFORT is set to 1, which means the operation does not compare placements of fill patterns. The operation places the center of the initial FILLSHAPE at the center of the area to be filled, then extends the fill outward as defined by STEP and OFFSET.

When EFFORT is greater than 1, the operation generates additional placements by placing the center of the initial FILLSHAPE at a different point within the *locus of possible initial placement locations*. This *locus of possible initial placement locations* is a rectangular area with its lower left corner in the center of the area to be filled with dimensions as follows:

- StrideX = FILLSHAPE extent in X direction + STEP
- StrideY = FILLSHAPE extent in Y direction + STEP

The *effort* defines the number of placements to try in the X and Y directions, as illustrated in [Figure 4-61](#).

**Figure 4-61. Relating Fill Extents to Effort**

Note that initial placements are snapped to the design grid.

Note that placement locations are limited by the design resolution, and snap to grid. Therefore the maximum number of placements tried is  $(\text{effort} * \text{effort})$ .

If you specify an EFFORT value greater than 1, it is recommended that you use a value of 4 for an optimal trade-off between performance and quality of results.

- [SPACE [EDGE] [INTERIOR] *space\_rule space\_layer*] ...

An optional keyword and argument set that specifies a layer used to identify the areas to be filled, and defines a spacing constraint (*space\_rule*) between fillshapes of the fill pattern and geometries on the layer. The *space\_layer* must be an original or derived polygon layer containing design geometry. This keyword and argument set may be specified more than once.

It is possible for multiple *space\_rules* to be defined for interactions between a fill shape and a specific layer; for example, one in a DFM Spec Fill Shape statement and one in a DFM Spec Fill statement. In this case, the more restrictive option is used:

For example, assume the following statements:

```
DFM SPEC FILL SHAPE ONE RECTFILL 0 0 1 1 SPACE .12 M1
DFM SPEC FILL SHAPE TWO RECTFILL 0 0 2 2 SPACE .24 M1

DFM SPEC FILL OUT
  FILLSHAPE
    OUTPUT "OUT1" "ONE"
    OUTPUT "OUT2" "TWO"
```

The combined multi-layer shape will have spacing to M1 greater than 0.24.

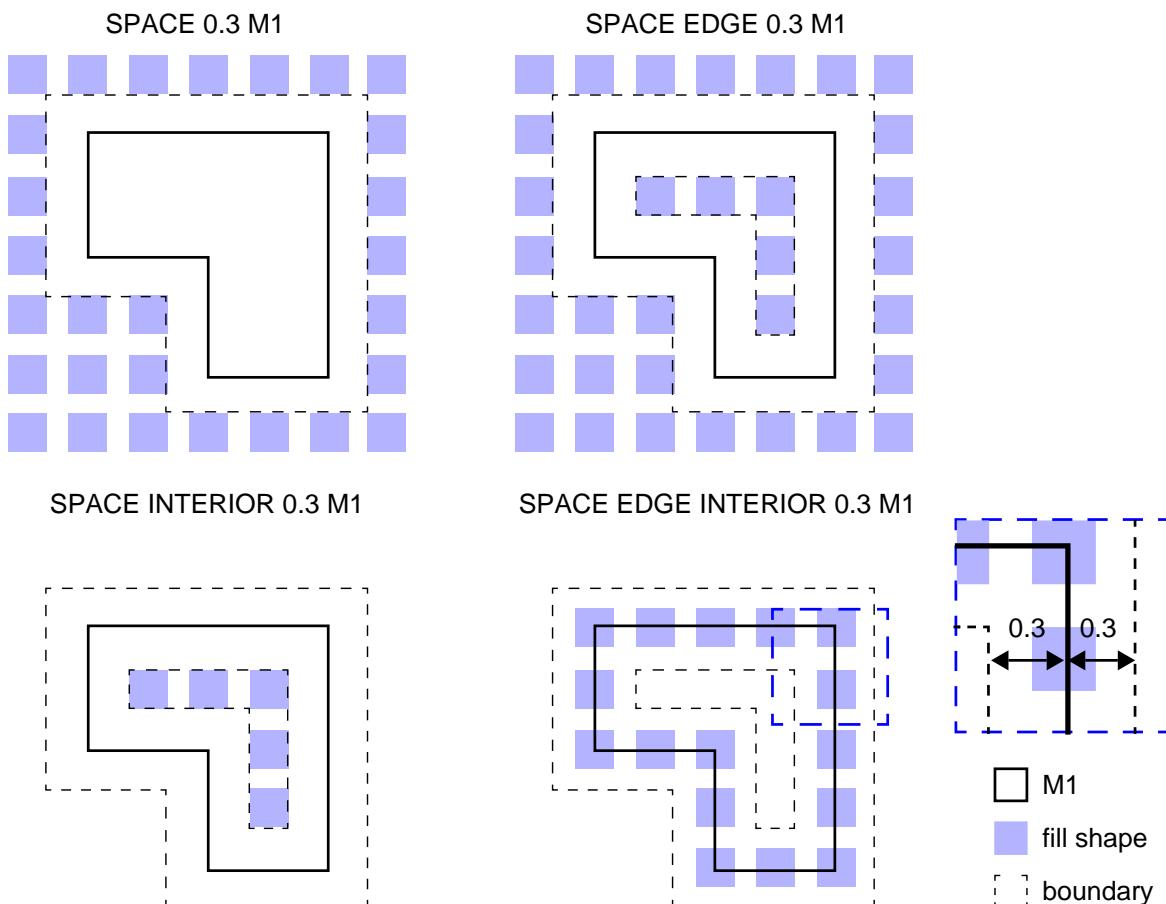
The optional EDGE keyword applies the specified *space\_rule* to the edges of the *space\_layer* instead of to the polygons. This allows placement of fill shapes both inside of and outside of polygons of *space\_layer*, but not interacting with the boundaries of these polygons.

The optional INTERIOR keyword indicates that fill shapes should be placed inside the polygons on *space\_layer* within the specified *space\_rule* distance to the boundaries of each polygon. This keyword is similar to INSIDE OF LAYER on a per-shape basis.

Specifying both EDGE and INTERIOR with SPACE effectively inverts the behavior of SPACE EDGE. That is, fill shapes are placed only within the specified *space\_rule* distance of the polygon boundaries on *space\_layer*.

Refer to [Figure 4-62](#).

**Figure 4-62. DFM Spec Fill SPACE Options**



- **ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN [constraint] OPPOSITE EXTENDED**

An optional keyword set that controls the metric for corner-to-corner spacing checks between fill regions.

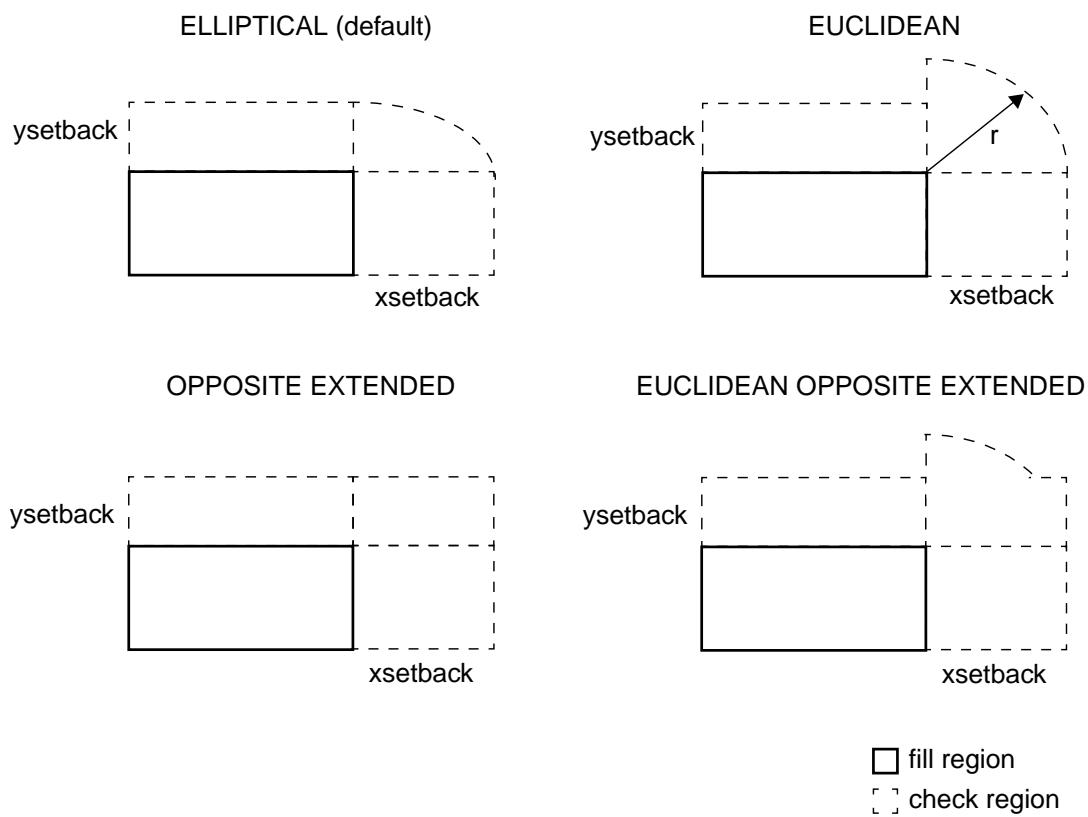
Fill patterns are generated on a grid for each fill region and are spaced at distances specified by the shape definition. However, in certain cases, generated shapes between regions can be spaced too close. Calibre performs a spacing check and resolves each conflict by eliminating one of the conflicting shapes. The distance between shapes is thus assured to be no less than the values specified by the STEP or SETBACK options. When AUTORotate is specified, the larger of the x and y values is used; without AUTORotate, the spacing check is different for the x and y directions. If this is not the desired behavior, the following options can be used to control the corner-to-corner spacing check metric (see also [Figure 4-63](#)):

- ELLIPTICAL — corner-to-corner distance is checked using the canonical equation of an ellipse:

$$x^2/xsetback^2 + y^2/ysetback^2 < 1$$

This is the default behavior.

- EUCLIDEAN [*constraint*] — corner-to-corner distance is checked with a radial arc with the largest of either *xsetback*, *ysetback* or the specified *constraint* value. Only less-than constraints are allowed. This option matches the definition of the nmDRC Euclidean metric.
- OPPOSITE EXTENDED — corner-to-corner distance is checked with an extension of the measurement region along each edge. The *xsetback* and *ysetback* values are used for the appropriate dimensions.
- EUCLIDEAN [*constraint*] OPPOSITE EXTENDED — corner-to-corner distance is checked using a combined check region created by OR-ing the EUCLIDEAN and OPPOSITE EXTENDED regions.

**Figure 4-63. Corner-to-Corner Spacing Metric Options (DFM Spec Fill)**

## Description

DFM Spec Fill is an SVRF statement that defines a fill specification for use with the [DFM Fill](#) operation.

The fill specification defines the shapes and sizes of one or more fill polygons or multi-layer polygon stacks. It also defines the polygon placements in terms of spacing constraints and defines the goals for fill generation in terms of *density\_constraint* and the optional *gradient\_constraint*. When generating the fill layer based on a fill specification, the DFM Fill operation creates polygons such that after their addition, the design satisfies both the user-specified *density\_constraint* and the user-specified *gradient\_constraint*.

Fill specifications can define the following:

- minimum density — the DFM Fill operation attempts to add fill to an area until this value is achieved. The goal is to add the minimum number of fill shapes needed to achieve this value.
- density gradient — the DFM Fill operation adjusts fill placement as needed to meet that constraint, limiting design density variation across the design.
- maximum density — the DFM Fill operation will not add fill to an area if adding that fill causes the density to exceed this value.

- fill patterns — you must define at least one fill pattern, specifying the fill shape, the spacing between shapes in the X and Y directions, and any offset required to stagger fill shapes. You may define as many fill patterns as you like, using smaller fill patterns to fill in between larger shapes as needed.

### **Analysis Windows Used for Adding Fill**

The DFM Fill operation adds fill as needed to achieve the goals specified through the constraints defined in the fill specification. When you define different values for WINDOW and GWINDOW, the operation manages window data by dividing the design data into *subwindows* that serve as the common grid that fits all of the different window options.

The size of the subwindows that form the common grid is as follows:

- $x = \text{the smallest of } wx, sx, gwx, gsx$
- $y = \text{the smallest of } wy, sy, gwy, gsy$

The restrictions imposed on arguments of WINDOW/GWINDOW options ensure that these subwindow dimensions produce a grid that is common to all WINDOW/GWINDOW options.

Additionally, the following checks are performed on fillshapes with respect to subwindows:

- The fillshape must fit inside the subwindow (that is, both the x and y dimensions of the fillshape are less than or equal to the x and y dimensions of the subwindow).
- At least one of the dimensions (x or y) of the fillshape must be less than or equal to half of the corresponding dimension of the subwindow. For example, if the x and y dimensions of the subwindow are both 2 units, either the x dimension of the fillshape must be less than or equal to 1 unit, or the y dimension of the fillshape must be less than or equal to 1 unit, or both.

Both of these conditions must be met; otherwise, an error is returned.

### **DFM Fill and Hierarchy Management**

Fill shapes are generated in the top level only. When viewing results, you may see some fill in pseudo cells. There can be two reasons for this:

- For performance reasons, the Calibre nmDRC Hierarchical Engine may inject some level of hierarchy into the results.
- If you have include the PSEUDO keyword in the DRC RESULTS DATABASE statement, saving the results automatically injects some hierarchy.

Note that all fill geometries are guaranteed to be intact, regardless of whether they are written out in the top level or not. That is, no geometries will be split at pseudo cell boundaries.

When the DFM Fill operation evaluates a design to see whether or not the density constraint is met, (and whether or not fill is required) the density analysis is performed hierarchically. Thus, geometries from the lower levels of hierarchy are considered in determining areas where fill can be applied.

Fill shapes are generated at the top level; however, shapes are pushed down into the hierarchy. Lower cells can therefore also have fill shapes.

Because the output of DFM Fill is a derived layer, users are free to manipulate this data as they see fit.

## Examples

### Example 1

Filling with a single fill pattern comprised of a rectangle.

```
DFM SPEC FILL Spec1 M1i [ ( AREA(M1i) + AREA(_FILL_) ) / AREA() ] \
> 0.30 GRADIENT < 0.5 ABSOLUTE
WINDOW 50
INSIDE OF EXTENT
WRAP
FILLSHAPE 0 0 10 10
STEP 3
SPACE 10 M1i
```

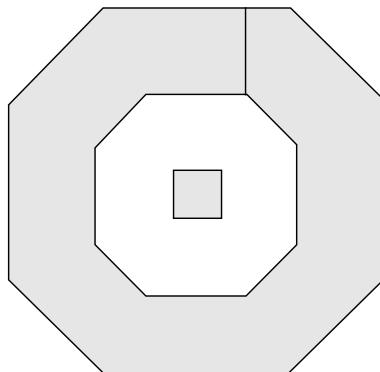
```
Example1fill = DFM FILL Spec1
```

### Example 2

Filling with a single fill pattern comprised of a donut shape polygon and a rectangle.

```
DFM SPEC FILL Spec2 metal2 metall1
FILLSHAPE
0.2 0 // Beginning of the first polygon (donut), exterior
.6 0
.8 .2
.8 .6
.6 .8
.2 .8
0 .6
0 .3
.2 .3 // 0-wide cut to get to interior of the polygon
.2 .5
.3 .6
.5 .6
.6 .5
.6 .3
.5 .2
.3 .2
.2 .3 // Interior is done, back to exterior via 0-wide cut
0 .3
0 .2
0.2 0
0.2 0 // Vertex is duplicate. End of the donut
0.35 0.35 // First vertex of the second polygon
0.35 0.45
0.45 0.45
0.45 0.35 // End of the second polygon.
// It will be auto-completed to 0.35 0.35
STEP .12 .12 EFFORT eff SPACE .060 metall1 SHAPESPACE m1_space
```

```
Example2fill = DFM FILL Spec2
```

**Figure 4-64. Fillshape for Example 2****Example 3**

Filling with multiple fill patterns.

```

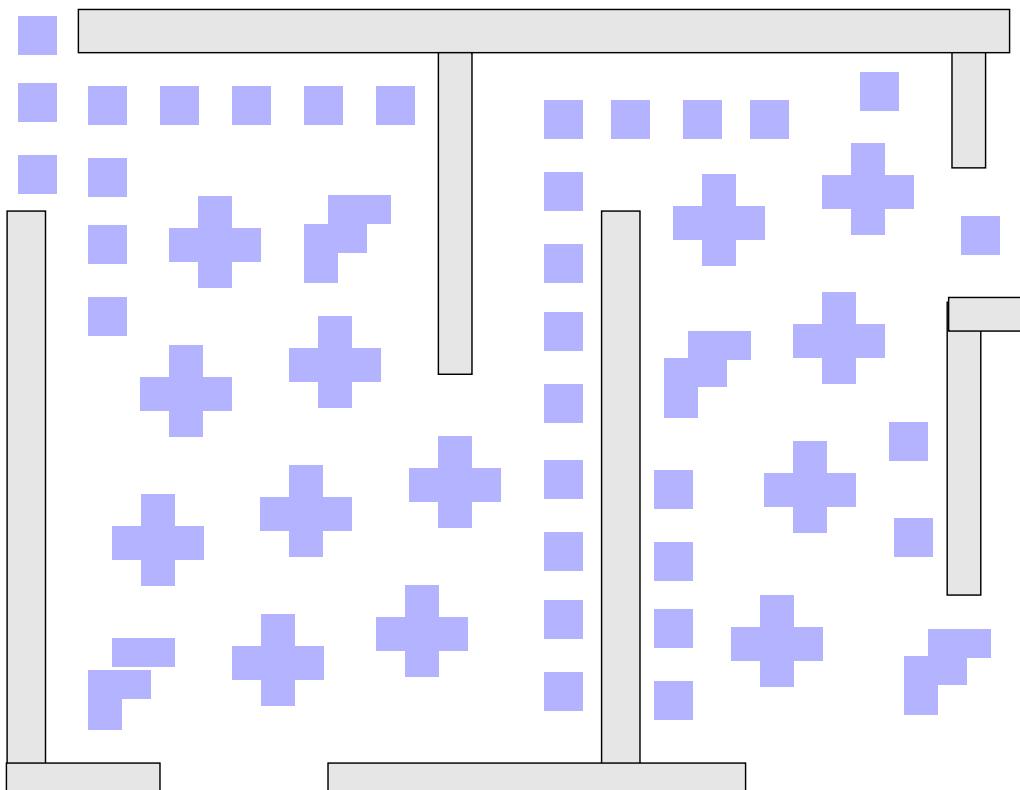
DFM SPEC FILL Spec3 metal2 metall1
FILLSHAPE .1 0
    .2 0
    .2 .1
    .3 .1
    .3 .2
    .2 .2
    .2 .3
    .1 .3
    .1 .2
    0 .2
    0 .1
    .1 .1
STEP .2 .2 OFFSET .1 .1 EFFORT eff SPACE .120 metall1 SHAPESPACE
m1_space
FILLSHAPE 0 0
    .1 0
    .1 .1
    .2 .1
    .2 .2
    .3 .2
    .3 .3
    .1 .3
    .1 .2
    0 .2
STEP .12 .12 EFFORT eff SPACE .060 metall1 SHAPESPACE m1_space
FILLSHAPE 0 0
    .1 0
    .1 .1
    .2 .1
    .2 .2
    .3 .2
    .3 .3
    .1 .3
    .1 .2
    0 .2
STEP .12 .12 EFFORT eff SPACE .060 metall1 SHAPESPACE m1_space

```

```
FILLSHAPE 0 0 .12 .12 STEP .12 .12 EFFORT 4 SPACE .060 metall1 SHAPESPACE
m1_space
FILLSHAPE 0 0 .12 .12 STEP .12 .12 EFFORT 4 SPACE .060 metall1
```

```
Example3fill = DFM FILL Spec3
```

**Figure 4-65. Fill for Example 3**



#### **Example 4**

Generate fill shapes in progressively smaller sizes. Split the output so that each fill shape goes into own layer for further processing.

```
m2fill_all = DFM FILL ONE
DFM SPEC FILL ONE metall2
[ ( AREA(metal2) + AREA(_FILL_) ) / AREA() ] > 0.25
WINDOW 50 50
FILLSHAPE OUTPUT "ONE" 0 0 .84 .84 STEP .12 .12
SPACE .360 metall2 SHAPESPACE 0.12
FILLSHAPE OUTPUT "TWO" 0 0 .84 .36 STEP .12 .12
SPACE .360 metall2 SHAPESPACE 0.12
FILLSHAPE OUTPUT "THREE" 0 0 .36 .84 STEP .12 .12 |
SPACE .360 metall2 SHAPESPACE 0.12
FILLSHAPE OUTPUT "FOUR" 0 0 .36 .36 STEP .12 .12
SPACE .360 metall2 SHAPESPACE 0.12
FILLSHAPE OUTPUT "FIVE" 0 0 .36 .12 STEP .12 .12
SPACE .360 metall2 SHAPESPACE 0.12
FILLSHAPE OUTPUT "SIX" 0 0 .12 .36 STEP .12 .12
SPACE .360 metall2
SHAPESPACE 0.12
```

```

FILLSHAPE OUTPUT "SEVEN" 0 0 .12 .12 STEP .12 .12
    SPACE .120 metal2 SHAPESPACE 0.12
FILLSHAPE OUTPUT "EIGHT" 0 0 .12 .12 STEP .12 .12 SPACE .120 metal2

m2fill11 = DFM FILL ONE ONE
m2fill12 = DFM FILL ONE TWO
m2fill13 = DFM FILL ONE THREE
m2fill14 = DFM FILL ONE FOUR
m2fill15 = DFM FILL ONE FIVE
m2fill16 = DFM FILL ONE SIX
m2fill17 = DFM FILL ONE SEVEN
m2fill18 = DFM FILL ONE EIGHT

```

**Example 5**

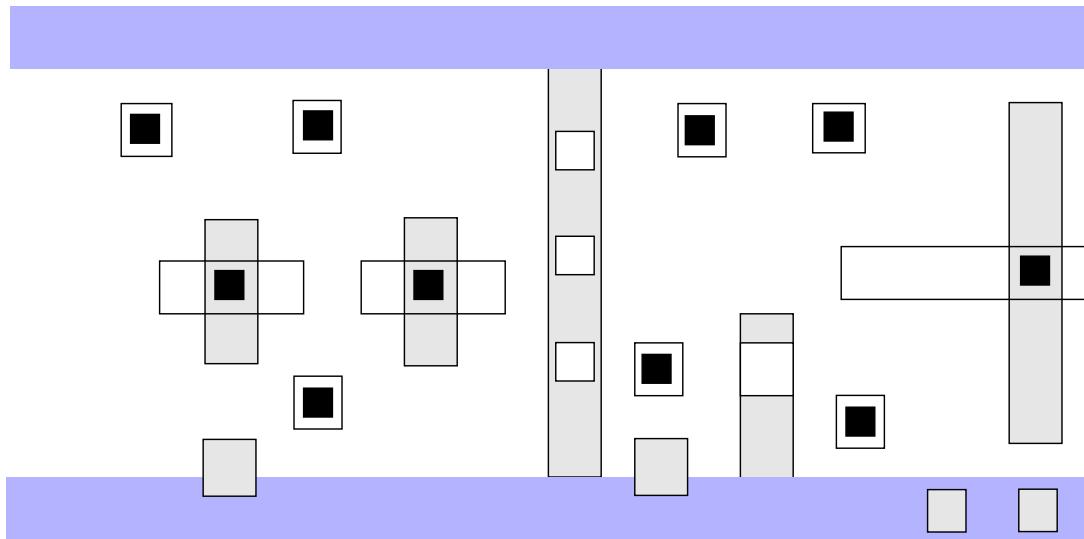
Generate simultaneous fill for metal1 and metal2, so that combined density of metal1 + fill, metal2 + fill and via1\_fill is greater than 0.36.

The fill shape is a cross. Metal1 fill stretches in Y-direction, Metal2 stretches in X-direction and via1 is just a square.

```

m1_fill     = DFM FILL ONE m1_fill
vial1_fill  = DFM FILL ONE vial1_fill
m2_fill     = DFM FILL ONE m2_fill
DFM SPEC FILL ONE metall1 metal2
[ ( AREA(metal1) + AREA(metal2) + AREA(_FILL_) ) / AREA() ] > 0.36
WINDOW 50
FILLSHAPE
    OUTPUT "m1_fill"    (-0.06) (-0.06) .06 .06 0 0.96 .12
    OUTPUT "vial1_fill" (-0.03) (-0.03) .03 .03
    OUTPUT "m2_fill"    (-0.06) (-0.06) .06 .06 0.96 0 .12
    STEP .12 .12
    SPACE .120 metall1
    SPACE .120 metal2

```



**Example 6**

This example shows how the FILLSTACK keyword and argument set can be used to simplify specification of a fill pattern with via stacks. Note that the complete DFM Spec Fill syntax is not shown for simplicity:

```
FILLSTACK UP MINSTACK 1
    OUTPUT "out_M1" "M1_F"
    OUTPUT VIA "out_V1" "V1_F"
    OUTPUT "out_M2" "M2_F"
    OUTPUT VIA "out_V2" "V2_F"
    OUTPUT "out_M3" "M3_F"
    OUTPUT VIA "out_V3" "V3_F"
    OUTPUT "out_M4" "M4_F"
    STEP ARY_S_X ARY_S_Y OFFSET ARY_O_X ARY_O_Y
```

This code is conceptually equivalent to specifying the following fill shapes with DFM Spec Fill (note that the STEP keyword is omitted for simplicity):

```
FILLSHAPE
    OUTPUT "out_M1" "M1_F"
    OUTPUT "out_V1" "V1_F"
    OUTPUT "out_M2" "M2_F"
    OUTPUT "out_V2" "V2_F"
    OUTPUT "out_M3" "M3_F"
    OUTPUT "out_V3" "V3_F"
    OUTPUT "out_M4" "M4_F"
FILLSHAPE
    OUTPUT "out_M1" "M1_F"
    OUTPUT "out_V1" "V1_F"
    OUTPUT "out_M2" "M2_F"
    OUTPUT "out_V2" "V2_F"
    OUTPUT "out_M3" "M3_F"
FILLSHAPE
    OUTPUT "out_M2" "M2_F"
    OUTPUT "out_V2" "V2_F"
    OUTPUT "out_M3" "M3_F"
    OUTPUT "out_V3" "V3_F"
    OUTPUT "out_M4" "M4_F"
FILLSHAPE
    OUTPUT "out_M1" "M1_F"
    OUTPUT "out_V1" "V1_F"
    OUTPUT "out_M2" "M2_F"
FILLSHAPE
    OUTPUT "out_M2" "M2_F"
    OUTPUT "out_V2" "V2_F"
    OUTPUT "out_M3" "M3_F"
FILLSHAPE
    OUTPUT "out_M3" "M3_F"
    OUTPUT "out_V3" "V3_F"
    OUTPUT "out_M4" "M4_F"
FILLSHAPE
    OUTPUT "out_M1" "M1_F"
FILLSHAPE
    OUTPUT "out_M2" "M2_F"
FILLSHAPE
    OUTPUT "out_M3" "M3_F"
FILLSHAPE
```

```
OUTPUT "out_M4" "M4_F"
```

Similarly, with DOWN specified:

```
FILLSTACK DOWN
OUTPUT "out_M1" "M1_F"
OUTPUT VIA "out_V1" "V1_F"
OUTPUT "out_M2" "M2_F"
OUTPUT VIA "out_V2" "V2_F"
OUTPUT "out_M3" "M3_F"
OUTPUT VIA "out_V3" "V3_F"
OUTPUT "out_M4" "M4_F"
```

this code is conceptually equivalent to the following FILLSHAPE definitions:

```
FILLSHAPE
OUTPUT out_M1 M1_F
OUTPUT out_V1 V1_F
OUTPUT out_M2 M2_F
OUTPUT out_V2 V2_F
OUTPUT out_M3 M3_F
OUTPUT out_V3 V3_F
OUTPUT out_M4 M4_F

FILLSHAPE
OUTPUT out_M2 M2_F
OUTPUT out_V2 V2_F
OUTPUT out_M3 M3_F
OUTPUT out_V3 V3_F
OUTPUT out_M4 M4_F

FILLSHAPE
OUTPUT out_M1 M1_F
OUTPUT out_V1 V1_F
OUTPUT out_M2 M2_F
OUTPUT out_V2 V2_F
OUTPUT out_M3 M3_F

FILLSHAPE
OUTPUT out_M3 M3_F
OUTPUT out_V3 V3_F
OUTPUT out_M4 M4_F

FILLSHAPE
OUTPUT out_M2 M2_F
OUTPUT out_V2 V2_F
OUTPUT out_M3 M3_F

FILLSHAPE
OUTPUT out_M1 M1_F
OUTPUT out_V1 V1_F
OUTPUT out_M2 M2_F

FILLSHAPE
OUTPUT out_M4 M4_F

FILLSHAPE
OUTPUT out_M3 M3_F

FILLSHAPE
OUTPUT out_M2 M2_F

FILLSHAPE
OUTPUT out_M1 M1_F
```

# DFM Spec Fill Optimizer

Specification statement

**DFM SPEC FILL OPTIMIZER** *optimizer\_name analyze\_layer1* [*analyze\_layer2 ...*]

‘[’ *optimization\_expression* ‘]’ [ *optimization\_limit* ]

[ MAGNITUDE *mag\_constraint* [ABSOLUTE | RELATIVE]]

[ GRADIENT *grad\_constraint* [ABSOLUTE | RELATIVE] [CORNER] ]

[ENVIRONMENT {> greater\_than\_value | < less\_than\_value |

< less\_than\_value > greater\_than\_value}]

Used only in Calibre YieldEnhancer applications.

## Summary

Defines a set of optimization criteria to be used in finding the optimal fill.

## Arguments

- ***optimizer\_name***

A required argument assigning a name by which this set of optimization criteria can be referenced.

- ***analyze\_layer1* [*analyze\_layer2 ...*]**

A required argument for use when performing any density calculations. This argument lists the name(s) of all layers referenced in the *optimization\_expression*. You must supply at least one *analyze\_layer*.

Note that these layers may or may not be the same as the layers used to constrain the placement of the generated fill, such as the space\_layer defined through DFM Spec Fill Shape or the DFM Spec Fill.

- ***optimization\_expression***

A required argument defining an expression that is used for optimization of fill generation. Typically, the *optimization\_expression* represents density.

The *optimization\_expression* may involve numbers (including numeric variables), [Unary Operators](#) (+, -, !, ~), [Binary Operators](#) (^, \*, /, +, -), parentheses ( ), variables, [Math Functions](#), [DFM Functions](#), [Conditional Expressions](#), and most [Measurement Functions](#).

The usual rules of operator precedence apply, and parentheses may be used to establish precedence in the calculation. In addition:

- The *expression* must be enclosed in brackets [ ].
- The temporary layer containing generated fill must appear in the expression at least once. In the *optimization\_expression*, you represent this temporary layer as “\_FILL\_”.
- Each input layer that appears in the expression must be defined as an *analyze\_layer*. (Note that \_FILL\_, the temporary layer containing generated fill, is not an input layer and therefore must not be defined as an *analyze\_layer*.)

- Division by 0 is defined not to satisfy any constraint.
- The *expression* cannot contain DFM Properties or [per-shape](#) operations.

### **Caution**



The value of the optimization\_expression must monotonically increase as fill shapes are added to the design. Note that this requirement is not checked at run time, but failure to conform to it will produce unexpected results (or no results at all).

In certain cases, an expression where the denominator approaches zero can cause calculations to be inaccurate and can lead to unexpected results. For example, consider the following two optimization expressions that consider exclusion zones:

$$[(\text{AREA}(\text{metal}) + \text{AREA}(\text{_FILL_})) / (\text{AREA}() - \text{AREA}(\text{EXCL}))] > \text{value}$$

$$[(\text{AREA}(\text{metal}) + \text{AREA}(\text{_FILL_}) + (\text{value} * \text{AREA}(\text{EXCL}))) / \text{AREA}()] > \text{value}$$

Even though the expressions are algebraically-equivalent, the first one can produce lower quality results (too many fill shapes) when  $\text{AREA}() - \text{AREA}(\text{EXCL})$  is very small.

- *optimization\_limit*

An optional argument defining a numeric constraint interpreted in user units. This constraint defines a goal that the geometry generation algorithm attempts to achieve, in terms of *optimization\_expression* values.

The *optimization\_limit* must be specified as a constraint. It can place an upper bound, a lower bound, or both on the *optimization\_expression* value.

- Lower bounds define a minimum *optimization\_expression* value that the operation attempts to achieve by adding fill until the minimum constraint is met.
- Upper bounds define an upper limit on the *optimization\_expression* value for an area to which fill is added. It is relevant only when GRADIENT is also specified, and used to ensure that the DFM Fill operation does not add too much fill to windows that are adjacent to a window with an extremely high original *optimization\_expression* value. When specified, fill will not be added if doing so would cause the *optimization\_expression* value to exceed the upper bound, even if this means that the gradient constraint is not met.

Note that ' $\geq$ ' and ' $>$ ' are treated as equivalent and mean ' $\geq$ '. Similarly, ' $\leq$ ' and ' $<$ ' are treated as equivalent and mean ' $\leq$ '.

The *optimization\_limit* is the highest-priority goal for generating fill. This means that if it is possible to achieve a *optimization\_expression* value within this specified constraint by adding geometry as defined by FILLSHAPE attributes, the geometry will be added.

- MAGNITUDE *mag\_constraint* [[ABSOLUTE](#) | [RELATIVE](#)]

An optional keyword and argument set used to define a maximum amount by which the *optimization\_expression* can vary over the total portion of the design being evaluated by the operation. That is, constraining the difference between the smallest *optimization\_expression* value and greatest *optimization\_expression* value in the design.

The primary argument, *mag\_constraint*, can be either of the following:

- *magnitude\_constraint* — a constant value constraint of the type less-than (<). The constant value must be a positive number, the value of which must meet the following criteria:
  - if *gradient\_constraint* is also specified, the constant value must be greater than the *gradient\_constraint*.
  - if a *optimization\_limit* with both a lower and upper bound is also specified, the constant value must be less than (*upper\_bound\_optimization\_limit* - *lower\_bound\_optimization\_limit*).
- *max\_magnitude\_constraint\_expression* — any expression that is supported by the DFM Analyze operation. When *max\_magnitude\_constraint\_expression* is used, it must be enclosed in square brackets, [ ]. For more information, refer to [DFM Expressions](#).

---

**Note**



*max\_magnitude\_constraint\_expression* can only be used with MAGNITUDE inside a DFM Spec Fill Optimizer statement. It cannot be used inside a DFM Spec Fill statement.

---

The secondary keywords, RELATIVE and ABSOLUTE define how the tool interprets the *mag\_constraint*:

- RELATIVE — the *mag\_constraint* is defined as a percentage. When specified:  
$$\text{mag\_constraint} = (\text{maximum\_density} - \text{minimum\_density}) / \text{maximum\_density}$$
which is equivalent to:  
$$\text{minimum\_density} = \text{maximum\_density} (1 - \text{mag\_constraint})$$
  - ABSOLUTE — the *mag\_constraint* is defined as an absolute value. (This is the default.) When specified, the tool attempts to add fill to achieve a minimum density such that:  
$$(\text{maximum\_density} - \text{mag\_constraint}) \leq \text{minimum\_density}$$
- GRADIENT *grad\_constraint* [ABSOLUTE | RELATIVE] [CORNER]
- An optional keyword and argument set, allowed only if *optimization\_expression* is specified, defining an optimal gradient to achieve between adjacent capture windows, where the gradient is a function based on the *optimization\_expression*. The operation will attempt to generate fill geometry to satisfy this constraint. This condition is *in addition to* the *optimization\_limit* criterion.
- The primary argument, *grad\_constraint*, can be either of the following:
- *gradient\_constraint* — a constant value constraint of the type less-than (<).

- *min\_gradient\_constraint\_expression* — any expression that is supported by the DFM Analyze operation. When *min\_gradient\_constraint\_expression* is used, it must be enclosed in square brackets, [ ]. For more information, refer to [DFM Expressions](#).

**Note**

 *min\_gradient\_constraint\_expression* can only be used with GRADIENT inside a DFM Spec Fill Optimizer statement. It cannot be used inside a DFM Spec Fill statement.

The gradient value of a window W is the maximum of

$$\{G(W, W_L), G(W, W_R), G(W, W_B), G(W, W_T)\}$$

where:

G is the gradient function

$W_L$  is the window immediately to the left of W

$W_R$  is the window immediately to the right of W

$W_T$  is the window immediately above W

$W_B$  is the window immediately below W

For windows A and B, where  $V_A$  is the *optimization\_expression* value for window A and  $V_B$  is the *optimization\_expression* value of window B the gradient function, G, is defined as:

- If RELATIVE is specified, or by default:

$$G(A,B) \equiv \frac{\|V_A\| - \|V_B\|}{\max(\|V_A\|, \|V_B\|)}$$

- If ABSOLUTE is specified:

$$G(A,B) \equiv \|V_A\| - \|V_B\|$$

In place of the *gradient\_constraint*, you may specify a *min\_gradient\_constraint\_expression* enclosed in square brackets.

If CORNER is specified with GRADIENT, then the set of windows in the GRADIENT calculation is extended to include the four windows touching W on each corner, which may not exist at certain window locations. GRADIENT value is reduced by  $\sqrt{2}$  for windows that are touching by a corner only.

**Note**

 If gradient values that are not as expected, check whether or not WRAP is used. Wrapped gradient windows are calculated based on the geometries on opposing edges of a design.

- ENVIRONMENT { $> greater\_than\_value$  |  $< less\_than\_value$  |  $< less\_than\_value > greater\_than\_value$ }

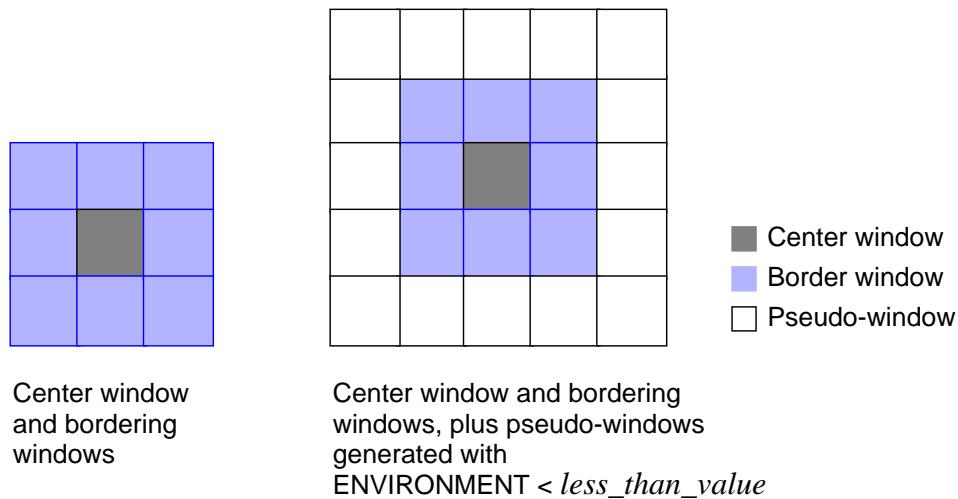
An optional keyword and argument set that controls the treatment of partial windows at borders of the design. Specifically, this keyword affects how partial windows on the right and upper edges of the design are treated in the optimization algorithm.

When the “ $>$ ” constraint is specified, any partial windows on the edges of the design are augmented to full size with an area density of the specified *greater\_than\_value*. When the “ $<$ ” constraint is specified, partial windows on the edges of the design are augmented to full size with area density of the specified *less\_than\_value*, and the gradient optimization algorithm assumes that areas surrounding the block have a density of the specified *less\_than\_value*. When both constraints are specified, partial windows on the edges of the design are augmented to full size with an area density of the specified *greater\_than\_value* and the gradient optimization algorithm assumes that areas surrounding the block have a density of the specified *less\_than\_value*.

When this keyword is not specified, partial windows, which are usually smaller than the desired capture window, can cause false positive errors during optimization.

When the ENVIRONMENT keyword is specified, partial windows are extended to full size for optimization purposes, and areas beyond design edges are assumed to have a particular density value. This value can be specified with the “ $>$ ” constraint for the keyword (that is, ENVIRONMENT  $> greater\_than\_value$ ).

The “ $<$ ” constraint of the keyword (that is, ENVIRONMENT  $< less\_than\_value$ ) allows you to define an optimization value for areas surrounding the design. This can be useful for filling design blocks prior to the assembly of the complete chip. In such scenarios, the maximum target density for the surrounding regions is known, and can be applied to the gradient optimization algorithm. For example, suppose that a design block has nine capture windows:



If the “ $<$ ” constraint is specified, then the optimization algorithm includes pseudo-windows surrounding the design block and assumes that their optimization value (or density) is equal

to the specified constraint value. In this way, you can control the gradient between border windows and geometries that may surround the design block being filled.

## Description

Defines a set of optimization criteria consisting of an optimization function and appropriate constraints. An optimizer should be viewed as an object or construct that can be instantiated multiple times through references in DFM Spec Fill or DFM Spec Fill Shape specifications.

Key points:

- When a fill shape referenced in a DFM Spec Fill has an optimizer explicitly specified for it through the DFM Spec Fill Shape statement that created it, the optimizer specified in the shape specification is used for optimization.
- When the DFM Spec Fill has an optimizer specified, this optimizer is used as an additional criteria to be met. That is, it is used in addition to any optimizers specified in the DFM Spec Fill Shape statements.
- All fill shapes in the same DFM Spec Fill that refer to the same OPTIMIZER will be optimized by the same instance of the optimizer. The shapes will be added in the order specified in DFM Spec Fill until the optimization constraint for each window is met.
- When multiple fill shapes referencing different optimizers are used to form a multi-layer fill pattern, all optimizers are used. The DFM Spec Fill keywords **FILLMIN** | **FILLMAX** control how many of the multi-layer patterns are actually added.
- FILLSHAPE/OUTPUT geometries from different DFM Spec Fills that refer to the same OPTIMIZER will be optimized independently. This means that fill generated by one DFM Spec Fill will not be counted in density calculation by another. If this behavior is not desired, then output of one DFM Spec Fill should be added to the original geometries as input to another DFM Spec Fill Optimizer.

## Case 1

No OPTIMIZERS specified in DFM Spec Fill Shape statements and:

```
DFM SPEC FILL X
    INSIDE OF LAYER new_layer
    WINDOW 5 5
    OPTIMIZER BASE
    FILLSHAPE
        OUTPUT metall1 "BIG_SQUARE"
        EFFORT 2
    FILLSHAPE
        OUTPUT metall1 "HORIZ_RECT"
        EFFORT 2
    FILLSHAPE
        OUTPUT metall1 "VERT_RECT"
        EFFORT 2
```

All shapes use same BASE as optimizer. First BIG\_SQUARE is added, then HORIZ\_RECT, then VERT\_RECT.

- BIG\_SQUARE shapes are factored into the OPTIMIZER values calculated for HORIZ\_RECT.
- BIG\_SQUARE and HORIZ\_RECT shapes are factored into the OPTIMIZER values calculated for VERT\_RECT.

### Case 2

No OPTIMIZERS specified in DFM Spec Fill Shape statements and:

```
DFM SPEC FILL X
    INSIDE OF LAYER new_layer
    WINDOW 5 5
    OPTIMIZER BASE
    FILLSHAPE
        OUTPUT metall1 "BIG_SQUARE"
        EFFORT 2
    FILLSHAPE
        OUTPUT metall1 "HORIZ_RECT"
        EFFORT 2
    FILLSHAPE
        OUTPUT metall1 "VERT_RECT"
        EFFORT 2
    FILLSHAPE
        OUTPUT metall1 "VERT_RECT"
EFFORT 4
```

As described in Case 1 plus:

- BIG\_SQUARE and HORIZ\_RECT and the first set of VERT\_RECT shapes are factored into the OPTIMIZER values calculated for the second set of VERT\_RECT shapes.

### Case 3

OPTIMIZER “EXTRA” specified for FILLSHAPE “BIG\_SQUARE” and:

```
DFM SPEC FILL X
    INSIDE OF LAYER new_layer
    WINDOW 5 5
    OPTIMIZER BASE
    FILLSHAPE
        OUTPUT metall1 "BIG_SQUARE"
        OUTPUT metall2 "HORIZ_RECT"
        EFFORT 2
    FILLMIN
```

Because the two OUTPUTs are supplied within the same FILLSHAPE, the two fill shapes are combined to form a 2-layer cell. Fill is generating by placing this cell in such a way as to satisfy at least one of the two optimizers, “EXTRA” or “BASE.”

# DFM Spec Fill Shape

Specification statement

```
DFM SPEC FILL SHAPE shape_name
{{RECTFILL x1 y1 x2 y2} |
 {POLYFILL x1 y1 x2 y2 [xn yn ...]} |
 {STRETCHFILL x1 y1 x2 y2 max_x max_y stretch_incr} }
[SPACE [EDGE] [INTERIOR] space_rule space_layer]
[SHAPESPACE [before_spacing] [after_spacing]]
[STEP {step | xstep ystep}]
[OFFSET {offset | xoffset yoffset}]
[SETBACK {setback | xsetback ysetback}]
[ON "metal_layer"]
[OPTIMIZER "optimizer_name"]
[UPDATE {ALL | "optimizer_name1" ["optimizer_name2" ...]}]
[ELLIPTICAL | EUCLIDEAN [constraint] | OPPOSITE EXTENDED | EUCLIDEAN
[constraint] OPPOSITE EXTENDED]
```

Used only in Calibre YieldEnhancer applications.

## Summary

Defines a fill shape and assigns it a name by which it can be referenced from within a DFM Spec Fill specification.

- When used for single layer shapes, this statement defines the fill shape in its entirety.
- When used for multi-layer shapes, this statement defines the portion of the shape that is drawn on a single (specific) layer.

## Arguments

- ***shape\_name***

A required argument assigning a name to this fill shape. You must specify a unique ShapeName for each DFM Spec Fill Shape statement.

- **RECTFILL { *x1 y1 x2 y2*}...**

A keyword used to define the fill pattern to be comprised of a single rectangle, as defined by the pair of vertices that follows. The operation uses these vertices as diagonally opposed vertices of a single rectangular fillshape.

You must specify one of: RECTFILL, POLYFILL, or STRETCHFILL.

- **POLYFILL {*x1 y1 x2 y2 [xn yn ...]*}**

A keyword used to define a fill pattern to be comprised of one or more polygons, as defined by the list of vertices that follow.

The operation uses each point as a vertex of a closed polygon. To define multiple polygons for POLYFILL, you must define each polygon in its entirety before defining subsequent

polygons. You indicate the completion of one polygon by repeating the final vertex. The vertex after a repeated vertex is treated as the beginning of a new polygon.

- The last vertex in a polygon may or may not match the first one. If it does not, the polygon will be automatically closed; the last vertex will be connected to the first one.
- Edges defined by the vertices must not intersect.
- Polygons with holes in them can be defined by using 0-wide cuts (edge-in is same as edge-out, but in the opposite direction). Refer to [Figure 4-64](#) for an example of this.
- When creating a polygon with holes in it, vertices of the polygon exterior should be enumerated in counter-clockwise direction, whereas inside holes should be enumerated in clockwise direction. This prevents cut lines from intersecting.

You must specify one of: RECTFILL, POLYFILL, or STRETCHFILL.

- **STRETCHFILL {*x1 y1 x2 y2 max\_x max\_y stretch\_incr* }**

A keyword used to define the fill pattern to be comprised of rectangles that are all variants of a user-defined minimum fill rectangle.

The arguments to this keyword are interpreted as follows:

- ***x1 y1 x2 y2***, defining the minimum fill rectangle,
- ***max\_x max\_y***, defining the direction the fill shape is allowed to stretch, and how far it can be stretched in that direction. Only one of the values can be non-zero. That non-zero value defines the maximum after-stretch length, in microns, of the fill shape in the associated direction.

For example:

- 0 0.6 — stretch fill shapes in the y direction, with maximum fill shape height of 0.6 microns.
- 0.6 0 — stretch fill shapes in the x direction, with maximum fill shape width of 0.6 microns.
- ***stretch\_incr***, defining an increment value to be added to a fill shape to stretch it. Together, ***stretch\_incr*** and the ***max\_x max\_y*** values define the number of variants produced:

$$(\text{max\_value} - \text{minimum\_rect\_dimension})/\text{stretch\_incr} + 1$$

where:

**max\_value** is the non\_zero value of {***max\_x max\_y***}

**minimum\_rect\_dimension** is the length of the minimum fill rectangle in the direction that the fill shape is allowed to stretch.

The total number of variants is limited to 40.

You must specify one of: RECTFILL, POLYFILL, or STRETCHFILL.

- SPACE [EDGE] [INTERIOR] *space\_rule space\_layer*

An optional keyword and argument set specifying the layers used to identify the areas to be filled, and defining a spacing constraint between fill of this fill pattern and geometries on the specified layer. The layer must be an original or derived polygon layer containing design geometry.

The optional EDGE keyword applies the specified *space\_rule* to the edges of the *space\_layer* instead of to the polygons. This allows placement of fill shapes both inside of and outside of polygons of *space\_layer*, but not interacting with the boundaries of these polygons.

The optional INTERIOR keyword indicates that fill shapes should be placed inside the polygons on *space\_layer* within the specified *space\_rule* distance to the boundaries of each polygon. This keyword is similar to INSIDE OF LAYER on a per-shape basis.

Specifying both EDGE and INTERIOR with SPACE effectively inverts the behavior of SPACE EDGE. That is, fill shapes are placed only within the specified *space\_rule* distance of the polygon boundaries on *space\_layer*.

Refer to [Figure 4-62](#).

- SHAPESPACE [*before\_spacing*] *after\_spacing*

An optional keyword and parameter set that defines the minimum spacing between the fill geometry generated for this fill pattern and geometry generated by FILLSHAPEs already added to the fill area (*before\_spacing*), and/or all subsequent FILLSHAPEs (*after\_spacing*). The effect of SHAPESPACE is to generate a keep-out area for other FILLSHAPEs. When not specified, SHAPESPACE defaults to the STEP value.

When two SHAPESPACE values affect relation of two fill shapes, the larger of the two space values is used. For example:

```
DFM SPEC FILL m1_fill
...
FILLSHAPE fs_1 ... SHAPESPACE 0.5
FILLSHAPE fs_2 ... SHAPESPACE 0.4
FILLSHAPE fs_3 ... SHAPESPACE 0.6 0.7
```

will result in the following spacing between fill shapes:

```
fs_1 to fs_2 spacing = .5
fs_1 to fs_3 spacing = .6 (larger of .6 and .5)
fs_2 to fs_3 spacing = .6 (larger of .6 and .4)
```

Even though the second SHAPESPACE value for fs\_3, 0.7, is required in order to specify the “before” spacing of 0.6, it is not used.

- STEP {*step* | *xstep ystep*}

An optional keyword defining the distance between adjacent shapes when repeating this fill shape to fill an area, as illustrated in the [Defining a Simple Array Fill Pattern](#) section in the *Calibre YieldAnalyzer and YieldEnhancer Reference Manual*. STEP accepts either one

argument, indicating that the STEP is the same for both the X and Y directions, or two arguments defining unique values for the STEP in the different directions. When not specified, the default step of 1 dbu is used. Using the default value of step is *not recommended*.

STEP is interpreted to be in the positive x and y directions.

- **OFFSET {offset | xoffset yoffset}**

An optional keyword defining the offsets between each shape and adjacent shapes, as illustrated in the [Defining a Staggered Array Fill Pattern](#) section in the *Calibre YieldAnalyzer and YieldEnhancer Reference Manual*. OFFSET accepts either one argument, indicating that the OFFSET is the same for both the X and Y directions, or two arguments defining unique values for the OFFSET in the different directions. When not specified, OFFSET is 0.

OFFSET is interpreted to be in the positive x direction and negative y direction.

When OFFSET is specified, you must consider the STEP and the OFFSET together to fully understand where the next shape will be added. Each fill shape is treated as if it were a cell, with a bounding box being the extent of all the polygons in the fillshape. The location of adjacent fillshape placements are defined by STEP and OFFSET plus the current position of the bounding box for the fillshape and the size of that bounding box. The operation calculates the location of the next fillshape as follows:

Assume (Xcurr, Ycurr) is the location of the current fillshape.

Coordinates of the next shape in the row (shape to the right):

```
Xnext = Xcurr + xstep + sizex  
Ynext = Ycurr - yoffset
```

Coordinates of the first shape in the next row (shape going up):

```
Xnext = Xcurr + xoffset  
Ynext = Ycurr + ystep + sizey
```

---

### **Caution**



The yoffset for DFM Spec Fill functions differently than for the SVRF [Rectangles](#) operation. If you are making the transition from a legacy SVRF-based fill solution to [DFM Fill](#), be sure to adjust the offset values accordingly.

---

- **SETBACK {setback | xsetback ysetback}**

---

### **Caution**



This option can have a significant impact on runtime and may produce DRC violations in your design. The use of this option is strongly discouraged except under special circumstances.

---

An optional keyword that defines the minimum spacing between fill shapes in adjacent (typically corner-to-corner) fill regions. In certain cases, fill shapes are spaced closer

together than the STEP value across these regions, which can cause DRC violations. By default, the post-filtering algorithm removes these violations. The SETBACK keyword allows you to control the behavior of the post-filtering algorithm.

SETBACK accepts either one argument, indicating that the value is the same for both the x and y directions, or two arguments, which allows you to specify unique values for the x and y directions. When SETBACK is not specified, the STEP values are instead used for the spacing checks in adjacent fill regions. Values specified with this keyword must be non-negative. A zero value effectively disables the post-filtering algorithm. If SETBACK is used with either STRETCHFILL or AUTORotate, only one value is allowed for both the x and y directions.

- [ON “*metal\_layer*”]

An optional keyword that allows you to specify how layers in 3-D fill are related to the original metal layers in the design, and in turn removes unnecessary area blocking by multi-layer fill shapes in multi-layer fill specifications. The *metal\_layer* argument specifies an original metal layer, and must be enclosed in quotes. This keyword should be used only for multi-layer fill specifications. If this keyword is used, it must be specified in all fill shapes referenced by DFM Spec Fill. Shapes with different ON identifiers must not be used in the same OUTPUT from DFM Fill.

- OPTIMIZER “*optimizer\_name*”

The name of the optimizer to use when evaluating the design for adding this fill shape.

- UPDATE {ALL | “*optimizer\_name1*” [“*optimizer\_name2*” ...]}

An optional keyword that, when specified, includes current fill shapes in the density calculations for the specified optimizers. If this keyword is not specified, only the optimizer specified with the OPTIMIZER keyword is updated.

This keyword can either list specific optimizers to be updated, or ALL may be used to indicate that all optimizers referenced by the current DFM Fill command are updated.

See also [Example 2](#).

- **ELLIPTICAL** | EUCLIDEAN [*constraint*] | OPPOSITE EXTENDED | EUCLIDEAN [*constraint*] OPPOSITE EXTENDED

An optional keyword set that controls the metric for corner-to-corner spacing checks between fill regions.

Fill patterns are generated on a grid for each fill region and are spaced at distances specified by the shape definition. However, in certain cases, generated shapes between regions can be spaced too close. Calibre performs a spacing check and resolves each conflict by eliminating one of the conflicting shapes. The distance between shapes is thus assured to be no less than the values specified by the STEP or SETBACK options. When AUTORotate is specified, the larger of the x and y values is used; without AUTORotate, the spacing check is different for the x and y directions. If this is not the desired behavior, the following options can be used to control the corner-to-corner spacing check metric (see also [Figure 4-66](#)):

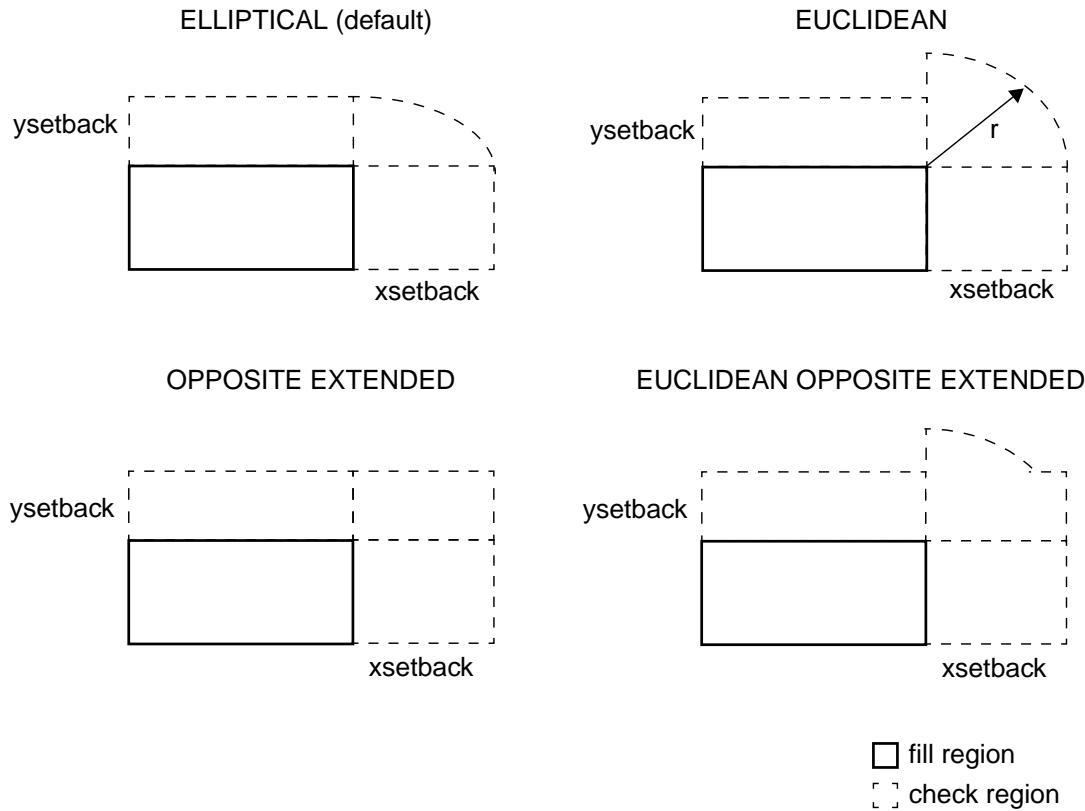
- ELLIPTICAL — corner-to-corner distance is checked using the canonical equation of an ellipse:

$$x^2/xsetback^2 + y^2/ysetback^2 < 1$$

This is the default behavior.

- EUCLIDEAN [*constraint*] — corner-to-corner distance is checked with a radial arc with the largest of either *xsetback*, *ysetback* or the specified *constraint* value. Only less-than constraints are allowed. This option matches the definition of the nmDRC Euclidean metric.
- OPPOSITE EXTENDED — corner-to-corner distance is checked with an extension of the measurement region along each edge. The *xsetback* and *ysetback* values are used for the appropriate dimensions.
- EUCLIDEAN [*constraint*] OPPOSITE EXTENDED — corner-to-corner distance is checked using a combined check region created by OR-ing the EUCLIDEAN and OPPOSITE EXTENDED regions.

**Figure 4-66. Corner-to-Corner Spacing Metric Options**



## Description

Defines a fill shape and assigns it a name by which it can be referenced from within a DFM Spec Fill specification.

- When used for single-layer shapes, this statement defines the fill shape in its entirety.
- When used for multi-layer shapes, this statement defines the portion of the shape that is drawn on a single (specific) layer.

Note that many of the fill shape options including SPACE, SHAPESPACE, STEP, OFFSET, and OPTIMIZER can also be specified in a DFM Spec Fill statement that references the fill shape. In this case the more restrictive option is used. For example, consider the following statements:

## Example

### Example 1

```
DFM SPEC FILL SHAPE ONE RECTFILL 0 0 1 1 SPACE .12 M1
DFM SPEC FILL SHAPE TWO RECTFILL 0 0 2 2 SPACE .24 M1

DFM SPEC FILL OUT
  FILLSHAPE
    OUTPUT "OUT1" "ONE"
    OUTPUT "OUT2" "TWO"
```

The combined multi-layer shape will have spacing to M1 greater than 0.24.

### Example 2

In the following example, we want large fill shapes to have target density of 30%, and small fill shapes to have a target density of 15%. If we don't specify UPDATE ALL for the large fill shape, the second fill shape could add too much fill, since it would not consider fill added by M2\_LARGE.

```
DFM SPEC FILL OPTIMIZER "OPT_1" M2
  [ ( AREA(M2) + AREA(_FILL_) ) / AREA() ] >= 0.3 <= 0.7

DFM SPEC FILL OPTIMIZER "OPT_2" M2
  [ ( AREA(M2) + AREA(_FILL_) ) / AREA() ] >= 0.15 <= 0.7

DFM SPEC FILL SHAPE "M2_LARGE"
  RECTFILL 0 0 2 2 STEP 0.2 OPTIMIZER "OPT_1" SPACE 0.2 M2 UPDATE ALL

DFM SPEC FILL SHAPE "M2_SMALL"
  RECTFILL 0 0 0.3 0.3 STEP 0.1 OPTIMIZER "OPT_2" SPACE 0.1 M2

DFM SPEC FILL Dummy_M2
  ...
  FILLSHAPE OUTPUT "out" "M2_LARGE"
  FILLSHAPE OUTPUT "out" "M2_SMALL"

DM2 = DFM FILL Dummy_M2
```

## DFM Spec Spatial Sample

Specification statement

**DFM SPEC SPATIAL SAMPLE** *spec\_name* [WINDOW *wx* [*wy*]] [NUMWIN *nx* [*ny*]]  
[RINDEX *r*] [CONVERGENCE *constraint*] [COVERAGE *constraint*]

Used only in Calibre YieldAnalyzer applications.

### Parameters

- *spec\_name*

Required argument that assigns a name by which the specification can be referenced in a [DFM Critical Area](#) operation.

- WINDOW *wx* [*wy*]

Optional keyword and argument set used to control the size of partitioned windows that are generated for the layout with DFM Critical Area SPATIAL SAMPLE. The *wx* and *wy* arguments specify the window dimensions in the x and y directions, respectively. If *wy* is not specified, the value of *wx* is used for it. This keyword cannot be used with NUMWIN.

- NUMWIN *nx* [*ny*]

Optional keyword and argument set used to control the number of partitioned windows for the layout with DFM Critical Area SPATIAL SAMPLE. The *nx* and *ny* arguments specify the number of windows in the x and y directions, respectively. If *ny* is not specified, the value of *nx* is used for it. This keyword cannot be specified with WINDOW. If neither WINDOW nor this keyword are specified, the values for *nx* and *ny* default to 64 for large designs. Note that if the layout extent is small or the defect size is large compared to the layout, *nx* and *ny* may default to 32, 16, or 8.

- RINDEX *r*

Optional keyword and argument that specifies which *defect\_radius* parameter the estimation function uses. The defect radii specified in the DFM Critical Area operation are ordered from smallest to largest. The radius index *r* starts at 0. If one or two *defect\_radius* parameters are specified, *r* defaults to 0. If more than two *defect\_radius* parameters are specified, *r* defaults to 1. This argument should reference the smallest particle radius that creates a non-empty critical area.

- CONVERGENCE *constraint*

Optional keyword and argument that defines a stopping point for calculation. The spatial sampling algorithm continues to calculate critical areas for representation sample sets until the convergence *constraint* is reached. The *constraint* must use “<” or “<=”. The default value is “<0.02” (less than 2%).

- COVERAGE *constraint*

Optional keyword and argument that instructs the tool to calculate a defined percentage of windows before the convergence constraint is taken into account. The constraint is defined as *windows* divided by *total\_windows*, and must use “>” or “>=”. For example, if you want

a coverage value of at least 50%, specify “ $\geq 0.5$ ” for the constraint. The default is no coverage requirement ( $\geq 0$ ). If the constraint is greater than 66%, the tool runs DFM Critical Area without spatial sampling to increase efficiency.

## Description

Defines a specification that can be used by [DFM Critical Area SPATIAL SAMPLE](#) to provide a rapid estimation of full-chip critical areas to a specified convergence constraint. Spatial sampling is best performed on large designs with a broad range of defect sizes. It is recommended that you do not use spatial sampling when you are more interested in accuracy than decreased runtime, or when you are interested in results from a BY CELL analysis.

## Examples

```
// defines a specification named spec_A with a coverage value of at
// least 2%
DFM SPEC SPATIAL SAMPLE spec_A COVERAGE >= 0.02
```

# DFM Spec Via Redundancy

Specification statement

## **DFM SPEC VIA REDUNDANCY** *vra\_spec\_name*

### **CONNECTIVITY**

*layer1 layer2 [layer3 ...]‘ ;’*

...

### **END CONNECTIVITY**

Used only in Calibre YieldAnalyzer applications.

## Parameters

- *vra\_spec\_name*

A required argument that specifies a name for a via redundancy analysis specification that can be referenced in a [DFM Redundant Vias](#) operation.

- *layer1 layer2 [layer3 ...]‘ ;’*

A required argument set that specifies a layer connectivity definition, which indicates that an electrical connection exists between overlapping shapes on the specified layers. Each layer in the definition must be a [Connect](#) layer if the definition is used with DFM Redundant Vias in HIERARCHICAL mode. At least two layers must be specified; the layers must be listed in the order that they occur in the layer stack (from bottom to top or from top to bottom).

Any number of unique connectivity definitions may be specified between the **CONNECTIVITY** and **END CONNECTIVITY** directives. Each definition must be terminated with a semicolon, which must be separated from the last layer defined with a space. In cases where the connectivity definitions are a subset of the complete Connect sequence, performance can be improved by using a reduced Connect sequence that includes only the layers specified for the connectivity definitions.

Layer definitions that have a common layer in the first position of one definition and in the last position of another definition can be joined together. For example, the following definitions:

```
CONNECTIVITY
    M1 VIA1 ;
    VIA1 M2 ;
    M2 VIA2 ;
    VIA2 M3 ;
END CONNECTIVITY
```

are equivalent to:

```
CONNECTIVITY
    M1 VIA1 M2 VIA2 M3 ;
END CONNECTIVITY
```

Layers that are concurrent in the stack (such as PO and OD) must be specified using separate definitions. In the following example, the three sets of definitions are equivalent.

```

CONNECTIVITY
PO CO M1 ;
OD CO M1 ;
M1 VIA1 M2 VIA2 M3 ;
END CONNECTIVITY

// or

CONNECTIVITY
PO CO M1 ;
OD CO M1 VIA1 M2 VIA2 M3 ;
END CONNECTIVITY

// or

CONNECTIVITY
PO CO M1 VIA1 M2 VIA2 M3 ;
OD CO M1 ;
END CONNECTIVITY

```

## Description

Defines a via redundancy analysis specification that can be referenced in a [DFM Redundant Vias](#) operation.

## Examples

```

CONNECT M1 M2 BY VIA1

DFM SPEC VIA REDUNDANCY m1m2_connectivity
CONNECTIVITY
    M1 VIA1 M2 ;
END CONNECTIVITY

m1m2_redundant_via1 = DFM REDUNDANT VIAS VIA1 m1m2_connectivity

```

# DFM Stamp

Layer operation

**DFM STAMP *input\_layer* BY *netID\_property***

## Parameters

- ***input\_layer***

Required argument that specifies the input layer. This layer must be either a polygon layer or a derived edge layer.

- ***netID\_property***

Required argument that specifies the name of a netID property on the input layer to be applied to the output layer. This argument must be enclosed in quotes ("").

## Description

Produces an output layer that is a copy of the input layer, preserving any attached DFM properties. Node numbers on the output layer are set by the ***netID\_property*** argument.

If the output layer does not require node numbers, only geometries and DFM properties are copied from the input layer.

## Examples

In this example, the node numbers on layer s are assigned using the netID properties from layer m. Layer s is then written to a DFM RDB file using the DFM RDB statement:

```
CONNECT metal2
metal2_p = DFM PROPERTY metal2 [N = NETID(metal2)]
m = DFM PROPERTY metal1 metal2_p OVERLAP [N = NETPROPERTY(metal2_p,N,1)]
s = DFM STAMP m BY "N"
write_db {
    DFM RDB s s.rdb NODAL
}
```

# DFM Summary Report

Specification statement

**DFM SUMMARY REPORT** *filename* [REPLACE | APPEND] [HIER]

## Parameters

- *filename*

Required argument that specifies the report file that is either created or appended to during execution.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in chapter 2, “Key Concepts”.

- REPLACE

Optional keyword that specifies to overwrite the previous contents of the summary file, if any, each time you run Calibre DFM. This is the default behavior if you do not specify this keyword.

- APPEND

Optional keyword that specifies to append to an existing summary report file. This parameter is useful in creating a log of runs in a single file.

- HIER

Optional keyword that instructs Calibre to create a section in the summary report that lists non-empty rule check statistics by layout database cell.

## Description

Enables creation of a file containing a summary report for a calibre -dfm run, which includes the following information:

- The date and time of the DFM run.
- The rule file pathname and title, if specified.
- The top-level layout cell pathname.
- The current working directory and user name.
- A summary of the number of original shapes processed per original layer.
- A summary of the number of results generated per rule check, as well as a listing of which rule checks from the rule file were not performed.
- All warnings generated.
- The total run time, result count, number of original shapes processed, and number of rule checks executed.

If you do not specify this statement, no summary file is generated.

## DFM Summary Report

---

You can specify the DFM Summary Report in Calibre Interactive — DFM by enabling **Write Summary Report File** in the **Outputs** tab.

### Examples

```
DFM SUMMARY REPORT dfm_report APPEND HIER
// write to dfm_report by appending it
```

# DFM Text

Layer operation

**DFM TEXT *layer* PROPERTY { STRING | NUMBER } *property\_name***

## Parameters

- ***layer***

A required argument that specifies the input layer. This layer must be an original layer.

- **STRING | NUMBER**

A required argument set that indicates the type of DFM property that is created from the text object. STRING indicates that a string-type property is created from the text object.

NUMBER indicates that a numeric-type property is created by converting the text value into a floating-point number. If this conversion fails, a warning is printed in the transcript, and no geometry is created.

- ***property\_name***

A required argument that specifies the name of the DFM property that is created. Note that DFM property names are case-sensitive.

## Description

Converts text objects to DFM properties. This operation creates a special unmerged polygon layer (since there are no limitations on how close two text objects can be together in the layout, the resulting marker polygons can overlap). The [DFM Property Merge](#) operation is the only operation that can use the unmerged layers as inputs and convert them into regular merged layers ([DFM RDB](#) can write out unmerged layers, and [DFM Copy](#) can copy these layers). When merging layers created by the DFM Text command, you must specify how to treat the overlapping text markers (refer to [Example 2](#) and [Example 3](#)).

## Examples

### Example 1

This example shows how to use DFM Text to create a string-type property, then access it using the SPROPERTY() function.

```
LAYER txt 1
TEXT LAYER txt // not necessary for text that is not connectivity
                // extraction text
txt_prop = DFM TEXT txt PROPERTY STRING s
merged = DFM PROPERTY MERGE txt_prop [ s = SPROPERTY(txt_prop,s,1) ]
```

### Example 2

This example shows how to detect locations where multiple text markers overlap and either suppress them from the output layer or create a separate layer.

```
merged_unique = DFM PROPERTY MERGE txt_prop
[s = SPROPERTY(txt_prop,s,1)]
[- = COUNT(txt_prop)] == 1
```

```
conflicts = DFM PROPERTY MERGE txt_prop
[- = COUNT(txt_prop)] > 1
```

### Example 3

This example shows how to collect all properties on the set of overlapping markers in a vector property.

```
text_prop_string = DFM TEXT txt1 PROPERTY STRING s
text_prop_number = DFM TEXT txt2 PROPERTY NUMBER s

merged_string = DFM PROPERTY MERGE text_prop_string
[ text = VSTRING(SPROPERTY(text_prop_string,s))]

merged_number = DFM PROPERTY MERGE text_prop_number
[ text = VECTOR(PROPERTY(text_prop_number,s))]
```

# DFM Transform

Layer operation

**DFM TRANSFORM** *layer* [ROTATION *angle* [CENTER *x\_center y\_center*]]  
 [FLIPX *y\_offset*] [FLIPY *x\_offset*]

Used only in Calibre YieldEnhancer applications.

## Parameters

- *layer*

Required argument that specifies the input layer. The input layer must be a polygon layer and is flattened prior to transformation.

- ROTATION *angle*

Optional keyword and argument pair that specify the angle of rotation in degrees. Positive values define a counter-clockwise rotation; negative values define a clockwise rotation.

- CENTER *x\_center y\_center*

Optional keyword and argument set that define the point of rotation in database units. If CENTER is not specified, the database origin is the center of rotation. This keyword can be used only in conjunction with ROTATION.

- FLIPX *y\_offset*

Optional keyword and argument set that flips (mirrors) the geometries about a horizontal line specified by *y\_offset*, which is specified in database units. If *y\_offset* is zero, geometries are flipped about the x-axis.

- FLIPY *x\_offset*

Optional keyword and argument set that flips (mirrors) the geometries about a vertical line specified by *x\_offset*, which is specified in database units. If *x\_offset* is zero, geometries are flipped about the y-axis.

## Description

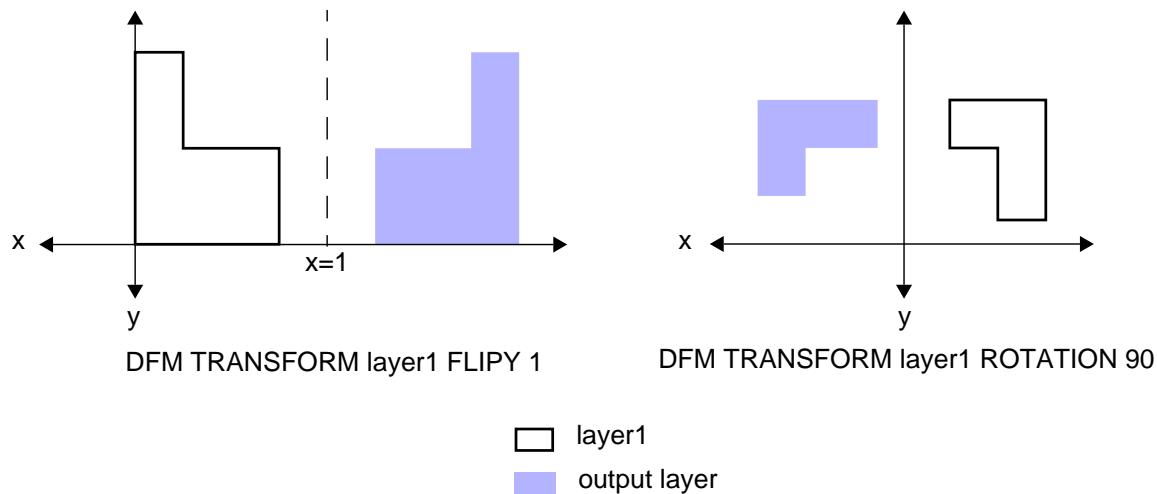
Transforms geometries on an input layer and produces an output layer containing the transformed geometries. Transforms include rotation about a specified point of origin and flipping (mirroring) across the x and y axes with a specified offset.

---

### Note



This operation is intended to be used in special cases such as metal fill replication or for other DFM enhancements on layouts containing several identical parts. It is not intended to be used as a part of regular layer derivations, since it creates flat output layers.

**Figure 4-67. DFM Transform Operation**

## Examples

### Example 1

```
DFM TRANSFORM M1_fill FLIPX 0
// flips geometries on layer M1_fill across the x-axis.
```

### Example 2

```
DFM TRANSFORM M1_fill ROTATION 90 CENTER 0.2 0.1
// rotates geometries on layer M1_fill 90 degrees counterclockwise
// about the point x = 0.2, y = 0.1.
```

# DFM Transition

Layer operation

```
DFM TRANSITION layer1 layer2 layer3 [NODAL]
  SPACE1 space1_value SPACE2 space2_value SPACE3 space3_value
  ENC1 enc1_value1 [enc1_value2] ENC2 enc2_value [enc2_value2]
  SIZE3 size3_value [START start_value]
  [OUTPUT1 | OUTPUT2 | OUTPUT3]
  [RDB_specification]
```

Used only in Calibre YieldEnhancer applications.

## Summary

DFM Transition is an SVRF rule file operation for improving via and contact transitions between two levels of interconnect. DFM Transition is part of the Calibre YieldEnhancer toolset. The operation is a polygon-directed, non-node-preserving layer constructor.

Due to internal considerations involving hierarchy management, DFM Transition cannot guarantee identical output between a flat run and a hierarchical run, nor can identical output from hierarchical runs be guaranteed across Calibre versions.

## Parameters

- ***layer1***  
A required original or derived polygon layer that is an interconnect layer.
- ***layer2***  
A required original or derived polygon layer that is an interconnect layer other than ***layer1***.
- ***layer3***  
A required original or derived polygon layer that is a via layer.
- **NODAL**  
Optional keyword that causes internally-generated nodal information to be written to the RDB. A node number is a property of the improved transition itself, not of the individual geometries in the improved transition. The node number of an (improved) transition is that of the original ***layer3*** shape in the transition. In the unlikely event that a START constraint other than == 1 is specified, and the ***layer3*** geometry in the transition collectively has more than one node number, then a node number is chosen arbitrarily.

Note that the DFM Transition operation remains a non-node-preserving layer constructor, regardless of the presence of the NODAL keyword. In other words, while its presence causes the operation to write nodal information to the RDB, the output (derived layer or rule check) from DFM Transition does not contain nodal information.

- **SPACE1** *space1\_value*

A required keyword and positive real number in user units that specify the minimum spacing between existing *layer1* polygons and new *layer1* polygons. New polygons generated from *layer1* must satisfy this spacing value.

- **SPACE2** *space2\_value*

A required keyword and positive real number in user units that specify the minimum spacing between existing *layer2* polygons and new *layer2* polygons. New polygons generated from *layer2* must satisfy this spacing value.

- **SPACE3** *space3\_value*

A required keyword and positive real number in user units that specify the minimum spacing between existing *layer3* polygons and new *layer3* polygons. New polygons generated from *layer3* must satisfy this spacing value.

- **ENC1** *enc1\_value1* [*enc1\_value2*]

A required keyword and positive real number in user units that specify the minimum enclosure of new *layer3* squares by *layer1* polygons. New polygons generated from *layer1* must satisfy this enclosure value. If *enc1\_value1* is specified alone, it pertains to the enclosure of all sides of new *layer3* squares. If *enc1\_value2* is specified, it pertains to one pair of opposite sides of new *layer3* squares, and *enc1\_value1* pertains to the other pair.

- **ENC2** *enc2\_value1* [*enc2\_value2*]

A required keyword and positive real number in user units that specify the minimum enclosure of new *layer3* squares by *layer2* polygons. New polygons generated from *layer2* must satisfy this enclosure value. If *enc2\_value1* is specified alone, it pertains to all sides of new *layer3* squares. If *enc2\_value2* is specified, it pertains to one pair of opposite sides of new *layer3* squares, and *enc2\_value1* pertains to the other pair.

- **SIZE3** *size3\_value*

A required keyword and positive real number in user units that specify the width of new squares on *layer3*.

- **START** *constraint*

An optional keyword and constraint that specify the number of existing vias that a transition must have in order for the operation to attempt to add new squares from *layer3*. The default is START == 1, if you do not specify this keyword.

- **OUTPUT1 | OUTPUT2 | OUTPUT3**

An optional keyword that specifies which layer the output comes from.

- **OUTPUT1** generates output from *layer1*
- **OUTPUT2** generates output from *layer2*
- **OUTPUT3** generates output from *layer3*

Output can come from only one layer per DFM Transition operation. The default, if you do not specify one of these keywords, is to output merged geometry from all three layers (new polygons may intersect original polygons).

You cannot specify any of these keywords if you specify RDB ONLY. Doing so generates a compiler error.

- *RDB\_specification*

An optional keyword and associated arguments and secondary keywords used to write results of this operation to a [Transitions RDB](#). The syntactical elements for RDB specification are:

```
RDB [ONLY] filename [MAXIMUM {value | ALL}] [NOEMPTY]
{{[WINDOW {w | x y}]} [STEP {s | x y}]}
[INSIDE OF {{x1 y1 x2 y2} | EXTENT | LAYER name}]] | 
[BY CELL [NOPSEUDO]]
```

When RDB is specified, the operation writes all transition geometries to the [Transitions RDB](#) specified by *filename*. Within the *filename*, the [Pattern Substitution](#) facility replaces all occurrences of the pattern “%\_t\_” with the name of the top cell for the design.

The OUTPUT keywords, which limit the *results* to transition data for a specific layer, do not affect the RDB; that is, the RDB generated by DFM Transition contains the same information, regardless of whether it is specified in an operation that returns the results for OUTPUT1, OUTPUT2, or OUTPUT3.

Also, if you include a DFM Transition operation multiple times, each differing only in terms of the OUTPUT, the RDB specification should only appear in one of the operations. If it appears in more than one operation, the transition data will be written to the RDB once for each appearance.

When ONLY is specified, results are sent only to the [Transitions RDB](#). No geometric data is sent to the usual nmDRC results database. RDB ONLY cannot be used with OUTPUT1, OUTPUT2, or OUTPUT3.

## Description

DFM Transition is an SVRF rule file operation for improving via or contact transitions between two levels of interconnect. DFM Transition is part of the Calibre YieldEnhancer toolset. The operation is a polygon-directed, non-node-preserving layer constructor.

You can use DFM Transition to insert redundant transition polygons into your layout to improve interconnect integrity. The DFM Transition operation inserts square vias into valid locations, according to conditions you specify. It also adds polygons as needed to the metal layers so that additional vias can be accommodated. Spacing and enclosure constraints for new polygons are part of the operation syntax.

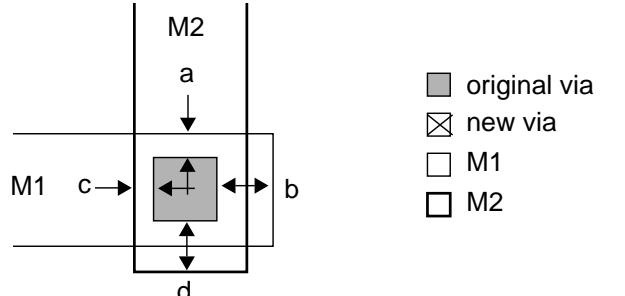
This operation attempts to add only one via at a time, at valid transition locations for each DFM Transition operation. This means multiple runs (or multiple operations in a single run) may be necessary to obtain the desired number of vias at transitions. If a via cannot be placed at a transition, that transition gets no new polygons.

When DFM Transition adds vias to the layout, the vias will be placed in one of 16 possible configurations. There are four configuration types and four orientations for each.

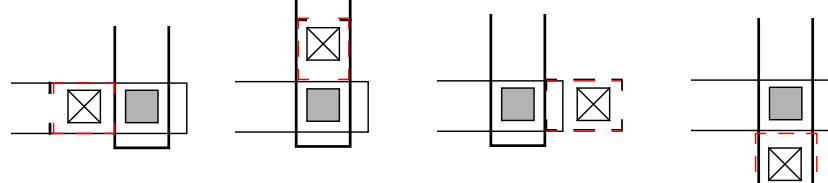
**Figure 4-68. Possible Via Configurations**

### Parameters

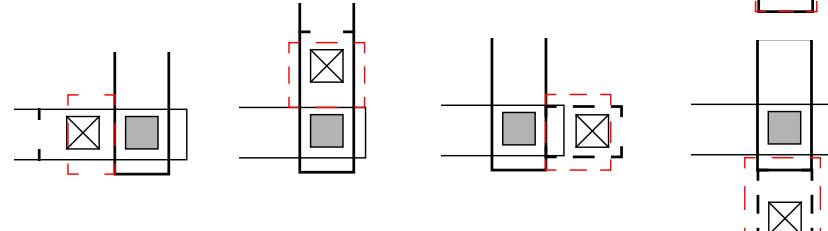
- V1 size, V1 space, M1 space, M2 space
- M1 encl (a), M1 ext(b),  
M2 encl(c), M2 ext(d)



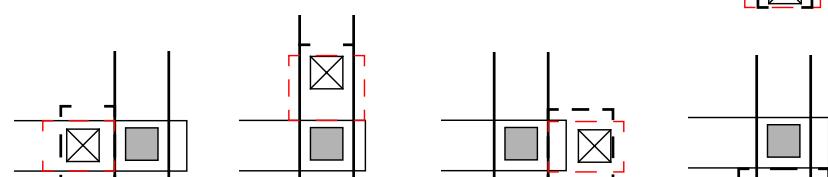
**pattern1-SS (skinny-skinny)**  
- small a & c enclosures  
- large b & d extensions



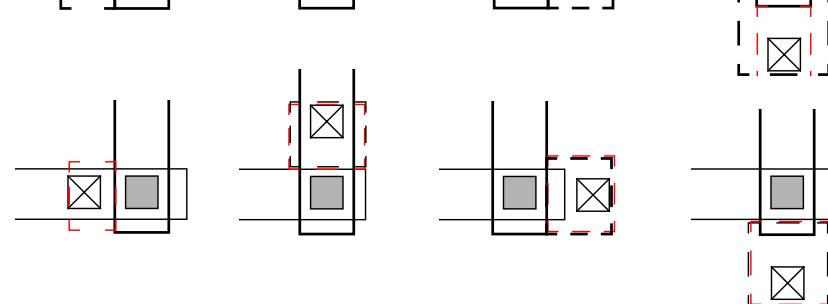
**pattern2-SF (skinny-fat)**  
- small a & large c enclosures  
- large b & small d extensions



**pattern3-FS (fat-skinny)**  
- large a & small c enclosures  
- small b & large d extensions



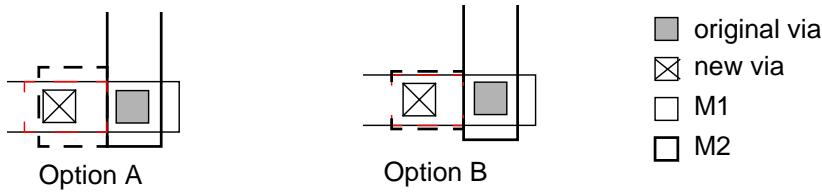
**pattern4-FF (fat-fat)**  
- large a & c enclosures  
- small b & d extensions



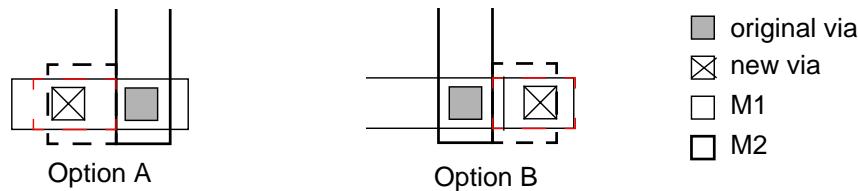
The operation works as follows:

1. For each transition, the operation attempts to add one additional square of *layer3*, with side length *size3\_value*. The operation does not move or replace existing *layer3* geometry.

2. To add the one additional square of *layer3*, the operation tries one of four locations. These locations are directly north, south, east, or west of one of the *layer3* polygons in the transition. The exact location and *layer3* polygon that is used is selected as follows:
- The operation examines all possible insertion options per single via (16 possible) to identify the nmDRC clean orientations for each cell type.
  - It chooses the option(s) that add the fewest jogs. For example, Option A adds fewer jogs than Option B:



- From the remaining options, it chooses the option(s) that adds the least total metal. For example, Option A adds less metal than Option B:



- If more than one option has the same amount of added metal, then the operation selects the placement based on the following preference:

Pattern1-SS > Pattern2-SF > Pattern3-FS > Pattern4-FF

where patterns are shown in [Figure 4-68](#).

- If there is more than one option left it will pick the via cell in order randomly.
- The polygons added consist of a *layer3* square of specified size, along with potentially new polygons from *layer1* or *layer2* that enclose the *layer3* square by a specified amount. All new polygons on all three layers satisfy minimum spacing and enclosure values defined in the operation, with both new and existing polygons as specified here:

**SPACE1:** New polygons on *layer1* satisfy external minimum spacing greater than or equal to *space1\_value*.

**SPACE2:** New polygons on *layer2* satisfy external minimum spacing greater than or equal to *space2\_value*.

**SPACE3:** New polygons on *layer3* satisfy external minimum spacing greater than or equal to *space3\_value*.

**ENC1:** New polygons in *layer1* enclose new squares on *layer3* by greater than or equal to *enc1\_value1*. If *enc1\_value2* is specified, then *layer1* encloses the new squares of *layer3* by *enc1\_value1* on two opposite sides, and *enc1\_value2* on the other two sides.

**ENC2:** New polygons in *layer2* enclose new squares on *layer3* by greater than or equal to *enc2\_value1*. If *enc2\_value2* is specified, then *layer2* encloses the new squares of *layer3* by *enc2\_value1* on two opposite sides, and *enc2\_value2* on the other two sides.

This part of the algorithm does not require the minimum width of *layer1* and *layer2*. The following is assumed:

minimum width in *layer1*  $\leq$  (*size3\_value* + min1)

where min1 is the lesser of *enc1\_value1* and *enc1\_value2*. Correspondingly, the following is also assumed:

minimum width in *layer2*  $\leq$  (*size3\_value* + min2)

where min2 is the lesser of *enc2\_value1* and *enc2\_value2*.

This assumption is necessary to ensure that new *layer1* and *layer2* polygons enclosing the added vias, when extended to touch the existing geometry, do not violate minimum width in those layers.

4. The output of this operation consists of all new polygons created for *layer1* if **OUTPUT1** is specified (the default), for *layer2* if **OUTPUT2** is specified, or for *layer3* if **OUTPUT3** is specified.

The NODAL keyword is used to generate connectivity information on *layer3* for various uses, primarily by DFM visualization and backannotation environments.

## RDB Output

RDB output is optional, but helpful in most cases where a detailed analysis of the results is desirable. RDB specification details and syntax are covered in the [Transitions RDB](#) section.

## Limitations

The actual nmDRC rules (including antenna rules) involving the input layers may be complex and extensive. This operation cannot take all of these factors into account. It is the user's responsibility to provide a simple parameter set that maintains DRC cleanliness over a potentially complex set of DRC relationships.

## Concurrency

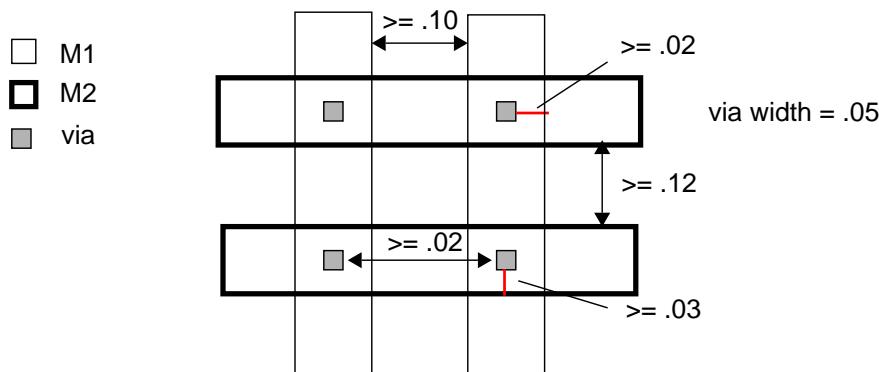
All DFM Transition operations with the same input layers (in the same order), and the same measurement parameters, are executed concurrently.

## Examples

### Example 1

Suppose you have the following design rule conditions:

- Minimum metal 1 spacing: 0.10 um
- Minimum metal 2 spacing: 0.12 um
- Minimum via spacing: 0.02 um
- Minimum enclosure of via by metal 1: 0.02 um
- Minimum enclosure of via by metal 2: 0.03 um
- Via width: 0.05 um



The following are example DFM Transition operations based on these design rules, which derive layers for M1, M2, and via polygons. These polygons can then be added to the original layout:

```

M1_new = DFM TRANSITION M1 M2 via
        SPACE1 0.10 SPACE2 0.12 SPACE3 0.02 ENC1 0.02 ENC2 0.03
        SIZE3 0.05 OUTPUT1

M2_new = DFM TRANSITION M1 M2 via
        SPACE1 0.10 SPACE2 0.12 SPACE3 0.02 ENC1 0.02 ENC2 0.03
        SIZE3 0.05 OUTPUT2

via_new      = DFM TRANSITION M1 M2 via
                SPACE1 0.10 SPACE2 0.12 SPACE3 0.02 ENC1 0.02 ENC2 0.03
                SIZE3 0.05 OUTPUT3
    
```

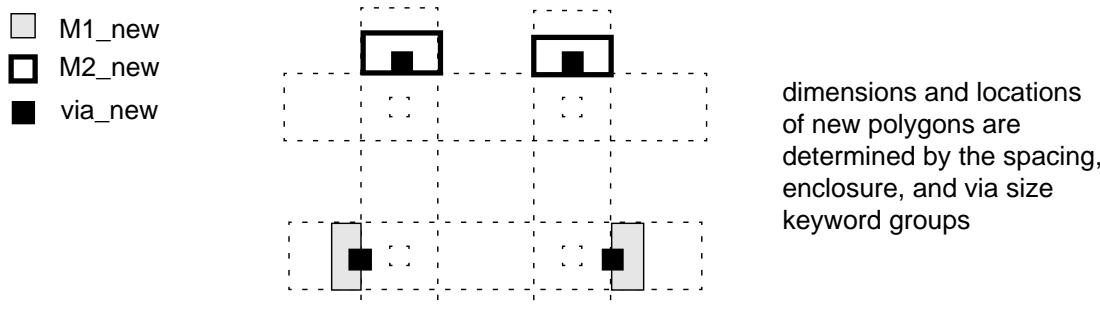
For each of these derivations, the keywords correspond to the design rule values, as follows:

- Minimum metal 1 spacing: SPACE1
- Minimum metal 2 spacing: SPACE2
- Minimum via spacing: SPACE3
- Minimum enclosure of via by metal 1: ENC1
- Minimum enclosure of via by metal 2: ENC2
- Via width: SIZE3

Any new polygons that are generated by the operation conform to these values.

The OUTPUT1, OUTPUT2, and OUTPUT3 keywords specify which layer the output comes from, and they follow the order of the layers specified in the DFM Transition operation. The layers M1\_new, M2\_new, and via\_new can subsequently be merged with the original layers by using an OR operation.

The placement of the vias is based upon internal algorithms, and is arbitrary. The tool attempts to place new vias north, south, east, or west of the location of the existing vias. The spacing, enclosure, and via width parameters determine where new polygons can be placed, and the dimensions of new polygons. No new vias are placed at transitions where a valid location cannot be found. The following figure shows how polygons might be generated for the previous example.



An additional keyword for the DFM Transition operation is the START keyword. You specify it with a constraint, and it instructs the DFM Transition operation to begin at transitions where the specified number of vias can be found. For instance, specifying START == 2 causes the operation to begin at transitions with two vias.

Continuing the previous example, here is how you could place successive vias at M1/M2 transitions:

```

M1_new = DFM TRANSITION M1 M2 via
        SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
        SIZE3 0.05 OUTPUT1

M2_new = DFM TRANSITION M1 M2 via
        SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
        SIZE3 0.05 OUTPUT2

via_new      = DFM TRANSITION M1 M2 via
                SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
                SIZE3 0.05 OUTPUT3

//add results to the original layers

next1 = M1 OR M1_new
next2 = M2 OR M2_new
next3 = via OR via_new

//add third via, where possible

M1_new2 = DFM TRANSITION next1 next2 next3
        SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
        SIZE3 0.05 START == 2 OUTPUT1
    
```

```

M2_new2 = DFM TRANSITION next1 next2 next3
          SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
          SIZE3 0.05 START == 2 OUTPUT2

via_new2    = DFM TRANSITION next1 next2 next3
          SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
          SIZE3 0.05 START == 2 OUTPUT3

//add results to previous

M1_final = next1 OR M1_new2
M2_final = next2 OR M2_new2
via_final = next3 OR via_new2

```

**Note**

The DFM Transition operation considers only spacing and enclosure constraints defined within the operation, and cannot account for all of the DRC conditions that need to be met in your design. You should always verify your modified design by using the standard nmDRC checks for all required layers. LVS and parasitic extraction verification may also be necessary after modifying interconnect layers.

**Example 2**

Consider these layer derivations:

```

M1_new = DFM TRANSITION M1 M2 via
          SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
          SIZE3 0.05 OUTPUT1

M2_new = DFM TRANSITION M1 M2 via
          SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
          SIZE3 0.05 OUTPUT2

via_new   = DFM TRANSITION M1 M2 via
          SPACE1 0.10 SPACE2 0.12 SPACE3 0.08 ENC1 0.02 ENC2 0.03
          SIZE3 0.05 OUTPUT3

```

The keywords correspond to the design rule values, as follows:

- Minimum metal 1 spacing: SPACE1
- Minimum metal 2 spacing: SPACE2
- Minimum via spacing: SPACE3
- Minimum enclosure of via by metal 1: ENC1
- Minimum enclosure of via by metal 2: ENC2
- Via width:SIZE3

Each of these derivations places M1, M2, and via polygons, respectively, at allowed locations in the layout. The OUTPUT keywords control which input layer is used for output. Only one via, maximum, is placed at a transition per run. Multiple runs may be necessary.

[DRC Check Map](#) is used for derived layer output.

## DFM Unselect Check

Specification statement

**DFM UNSELECT CHECK** *rule\_check* [*rule\_check* ...]

### Parameters

- *rule\_check*

A required argument that specifies the name of a rule check or rule check group. You must specify at least one *rule\_check*.

### Description

Allows you to specify DFM rule checks that are not executed when Calibre is run in -dfm mode.

See also [DFM Select Check](#).

### Examples

```
// excludes check1 and check3 from being executed in a Calibre -dfm run
DFM UNSELECT CHECK check1 check3

// executes only check3 in a Calibre -dfm run
DFM SELECT CHECK check1 check3
DFM UNSELECT CHECK check1
```

## DFM YS Autostart

Specification statement

**DFM YS AUTOSTART *TVF\_function tcl\_proc [tcl\_proc ...]***

### Parameters

- ***TVF\_function***

A required argument that specifies the name of a TVF Function defined in the rule file.

- ***tcl\_proc [tcl\_proc ...]***

A required argument that specifies the name of one or more Tcl procedures defined inside the TVF Function. The Tcl procedures are executed in the order that they are specified.

### Description

Allows Tcl code within a rule file to be executed automatically after a calibre -dfm run, or prior to a YieldServer run. If an SVRF rule file has the statement DFM YS Autostart and Calibre is invoked with the -autostart option, the Tcl functions listed after the DFM YS Autostart statement will be executed in the order in which they are specified.

### Examples

The following code, when included in a rule file, lists the vertices for geometries on layer met1. It can be executed by running “calibre -dfm -hier *rule\_file* -ys -autostart” at the command line. Alternatively, if the DFM database already exists, you can execute the code by running “calibre -ys -dfmdb *dfmdb* -autostart” at the command line.

```
DFM YS AUTOSTART sample_function
    list_vertices
    exit_ys

TVF FUNCTION sample_function /*

    proc list_vertices {} {
        set layer met1
        set iter [dfm::get_geometries $layer]
        puts "Vertices for geometries on layer $layer:\n"
        while { $iter ne "" } {
            set vertices_list [dfm::get_data $iter -vertices]
            puts "$vertices_list\n"
            dfm::inc iter
        }
    }

    proc exit_ys {} {
        puts "Exiting"
        exit -force
    }
*/ ]
```

## Disconnect

Connectivity operation

### DISCONNECT

Used only in nmDRC applications using incremental connectivity.

---

**Note**

This statement is *rarely needed and should be avoided* in most rule files. See “[Using Disconnect](#)” in the *Calibre Verification User’s Manual* for additional discussion.

---

### Description

Allows the total deletion of an existing connectivity model in an incremental [Connect](#) sequence. Incremental Connect sequences occur when using the [DRC Incremental Connect YES](#) specification statement in the rule file.

Use of Disconnect can result in unnecessary performance penalties. In general, a well-designed connectivity sequence is preferable and achievable in the majority of rule files. If you believe you need to use Disconnect, you should discuss your methodology with a customer support engineer first.

The Disconnect statement causes the currently built-up connectivity to be discarded; the next Connect operation causes the connectivity model to be built up from scratch. Note, this does not apply to text attachment performed with [Attach](#) statements.

A Disconnect operation does not begin a new connectivity zone, rather, the presence of a Disconnect statement in a connectivity zone causes all connectivity to be deleted prior to the execution of Connect statements in following connectivity zones. Appropriate compile-time modifications of connectivity algorithms for layers used in incremental connectivity are applied. For example, connectivity of a layer cannot be verified across a Disconnect statement.

You can specify this statement any number of times.

Disconnect is ignored in non-nmDRC applications and in nmDRC applications where the DRC Incremental Connect YES specification statement is not specified. Disconnect is meaningless in a non-incremental Connect flow because all Connect operations are run as a single block.

### Examples

This example shows the general idea of using Disconnect. Remember, Disconnect is useful only in *rare situations for advanced checks* where there is no alternative to discarding the existing connectivity model. This does not generally apply to most antenna checks.

```
DRC INCREMENTAL CONNECT YES  
  
CONNECT layerA layerB // first connectivity zone  
...  
// layer operations  
...  
CONNECT layerC layerD // second connectivity zone  
...
```

```
// layer operations
...
DISCONNECT           // delete all connectivity and start over

CONNECT layerE layerF // new initial connectivity zone
...
```

## Donut

Layer operation

### DONUT layer [constraint]

#### Parameters

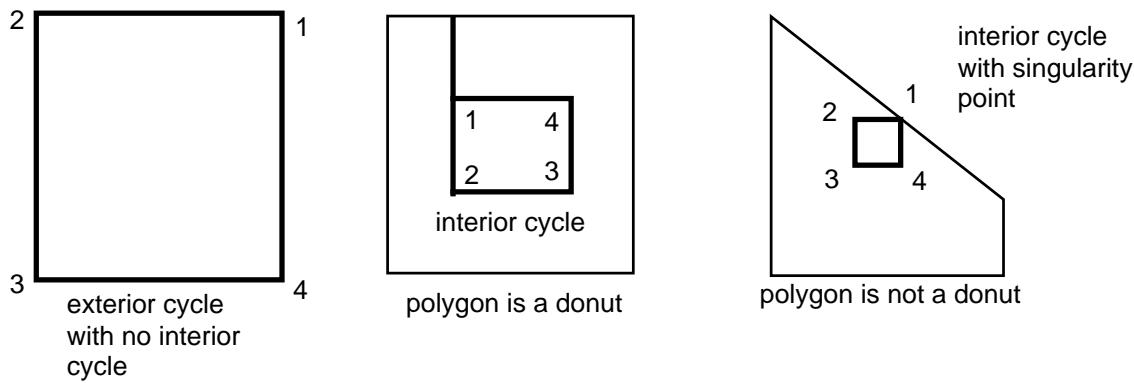
- *layer*  
A required original layer or layer set, or a derived polygon layer.
- *constraint*  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. It specifies the number of interior cycles a *layer* polygon must have to be selected.

#### Description

Selects all *layer* polygons having interior cycles (holes) that are fully enclosed within a *layer* polygon. If used, the *constraint* specifies the number of interior cycles a polygon must have in order to be selected.

The tool divides the vertices (or edges) of any merged polygon into one or more non-intersecting cycles, of which there are two types: exterior and interior. The cycle containing the right-most vertex is called the exterior cycle. A merged polygon always has only one exterior cycle. The remaining cycles are the interior cycles. An interior cycle consists of those vertices forming a hole within the merged polygon area. A merged polygon may or may not have interior cycles. Figure 4-69 illustrates the concept of cycles.

**Figure 4-69. Exterior and Interior Cycles**



An interior cycle that touches the perimeter of the polygon at one point (called a singularity point) causes the polygon not to satisfy the operation. In other words, it is not considered a Donut hole.

See also [Not Donut](#), [Holes](#).

## Examples

### Example 1

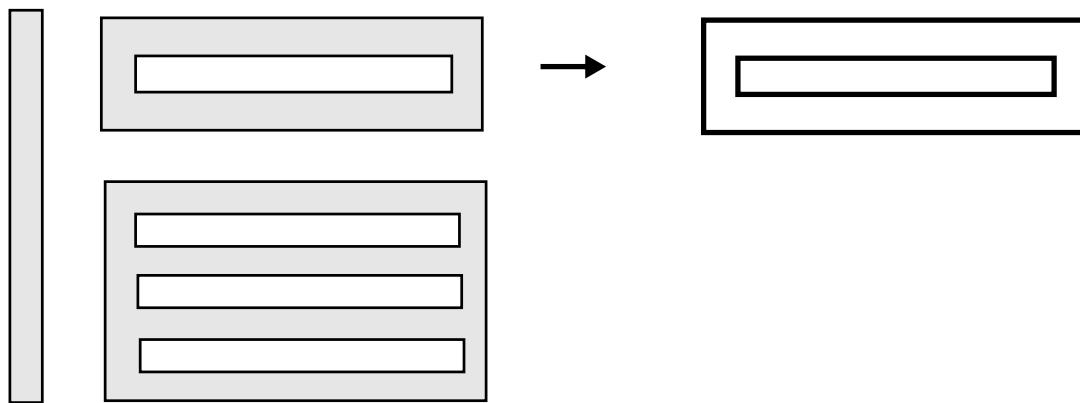
```
closed_metal { @ Closed structures are not allowed on the metal layer.  
metal DONUT  
}
```

### Example 2

Figure 4-70 shows a Donut operation corresponding to the following rule check.

```
donut_metal {  
DONUT metal <= 2  
}
```

**Figure 4-70. Donut**



## Drawn Acute

Layer operation

### DRAWN ACUTE

#### Description

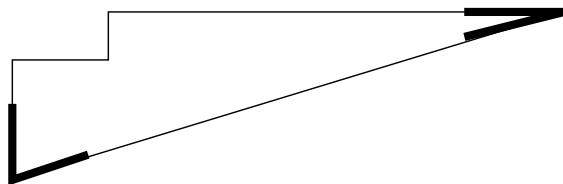
Selects acute angles on original shapes read from the layout database during a nmDRC-style application run. This operation produces a derived error layer indicating the acute angles in the layout database. Each element of the derived error layer is a cluster of two edges; the two edges are the adjacent edges of the original geometry whose common endpoint is the vertex point of the acute angle. The output edge cluster is trimmed to a distance of one user unit from the location of the acute angle.

The operation only reads original layers required by the check set. The check set does not necessarily include the entire set of original layers referenced in the rule file, as only the layers actually used in layer operations are read in.

This operation only acts on unmerged original data and does not use an input layer parameter. The Drawn Acute operation is error-directed, which means it cannot generally be used to derive layers. See “[Derived Error Layers](#)” in the *Calibre Verification User’s Manual*.

The polygon shown in Figure 4-71 has two pairs of highlighted edges that comprise selected acute angles. See also [Flag Acute](#), [Exclude Acute](#), [Drawn Angled](#), [Drawn Skew](#), and [Angle](#).

**Figure 4-71. Shown Acute Angles**



#### Examples

```
acute_angle_check {
    DRAWN ACUTE
}
// does not check layers that do not appear in other rule
// checks used in the run
```

# Drawn Angled

Layer operation

## DRAWN ANGLED

### Description

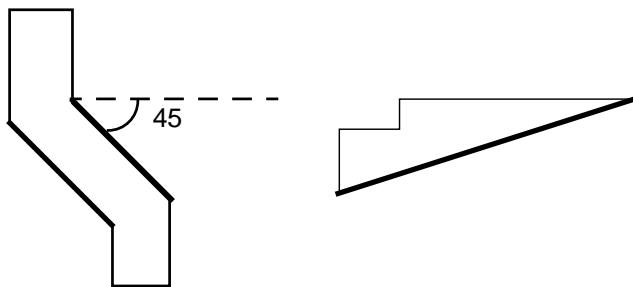
Selects all edges that are not orthogonal to the database axes on original shapes read from the layout database during a nmDRC-style application run. This includes 45-degree edges, which the [Drawn Skew](#) operation does not select. This operation produces a derived error layer indicating the angled edges in the layout database.

The operation only reads original layers required by the check set. The check set does not necessarily include the entire set of original layers referenced in the rule file, as only the layers actually used in layer operations are read in.

This operation only acts on unmerged original data and does not use an input layer parameter. The Drawn Angled operation is error-directed, which means it cannot generally be used to derive layers. See “[Derived Error Layers](#)” in the *Calibre Verification User’s Manual*.

The polygon shown in Figure 4-71 has highlighted edges that are angled angles. See also [Flag Angled](#), [Exclude Angled](#), [Flag Skew](#), [Angle](#), and [Deangle](#).

**Figure 4-72. Angled Output**



### Examples

```
angled_check {
    DRAWN ANGLED
}
// does not check layers that do not appear in other rule
// checks used in the run
```

## Drawn Offgrid

Layer operation

### DRAWN OFFGRID

#### Description

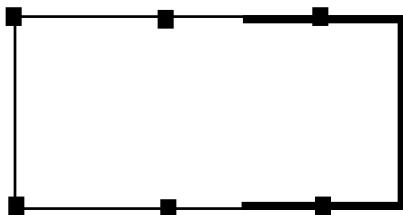
Selects off-grid vertices on original shapes read from the layout database during a nmDRC-style application run. The nmDRC system uses the [Resolution](#) or [Layer Resolution](#) specification statement in the rule file to determine what constitutes an off-grid vertex. This operation produces a derived error layer indicating the off-grid vertices in the layout database. Each element of the derived layer is a cluster of two edges; the two edges are adjacent edges of the original geometry whose common endpoint is the off-grid vertex. The output edge cluster is trimmed to a distance of one user unit from the location of the vertex.

This operation only reads original layers required by the check set. The check set does not necessarily require the entire set of original layers referenced in the rule file, as only the layers actually used in layer operations are read in.

This operation only acts on unmerged original data and does not use an input layer parameter. The Drawn Offgrid operation is error-directed, which means it cannot generally be used to derive layers. See “[Derived Error Layers](#)” in the *Calibre Verification User’s Manual*.

The polygon shown in Figure 4-73 has two pairs of highlighted edges which include two off-grid vertices.

**Figure 4-73. Off-Grid Vertices**



See also [Flag Offgrid](#), [Exclude Offgrid](#), [Offgrid](#), and [Snap Offgrid](#).

#### Examples

```
offgrid_vertex_check {
    DRAWN OFFGRID
}
// does not check layers that do not appear in other rule
// checks used in the run
```

## Drawn Skew

Layer operation

### DRAWN SKEW

#### Description

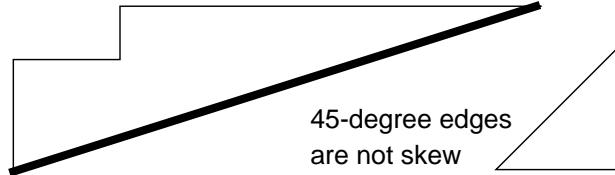
Selects skew edges on original shapes read from the layout database during a nmDRC application run. Skew edges are edges not orthogonal to the coordinate axes and do not have a slope of +1 or -1. This operation produces a derived error layer that indicates the skew edges in the layout database. Each element of the derived layer is a cluster of one edge, which is the skew edge.

This operation only reads original layers that are required by the check set. The check set does not necessarily require the entire set of original layers referenced in the rule file, as only the layers actually used in layer operations are read in.

This operation only acts on unmerged original data and does not use an input layer parameter. The Drawn Offgrid operation is error-directed, which means it cannot generally be used to derive layers. See “[Derived Error Layers](#)” in the *Calibre Verification User’s Manual*.

The polygon shown in Figure 4-74 has one skew edge that is highlighted. See also [Flag Skew](#), [Exclude Skew](#), [Drawn Angled](#), [Drawn Acute](#), [Angle](#), and [Deangle](#).

**Figure 4-74. Skew Edge**



#### Examples

```
skew_edge_check {
    DRAWN SKEW
}
// does not check layers that do not appear in other rule
// checks used in the run
```

## DRC Boolean Nosnap45

Specification statement

### **DRC BOOLEAN NOSNAP45 {NO | YES}**

Used only in Calibre nmDRC-H.

#### Parameters

- **NO**

Keyword that specifies 45-degree edges must snap, resulting in a merged shape with slightly non-45-degree edges even though the original input was octagonal. This is the default behavior when you do not include this statement in the rule file.

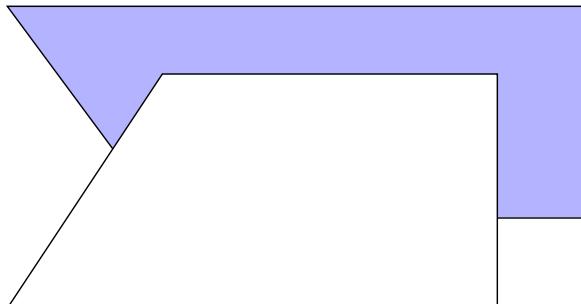
- **YES**

Keyword that specifies the merged shape must remain octagonal at the expense of inserting a 1 dbu edge.

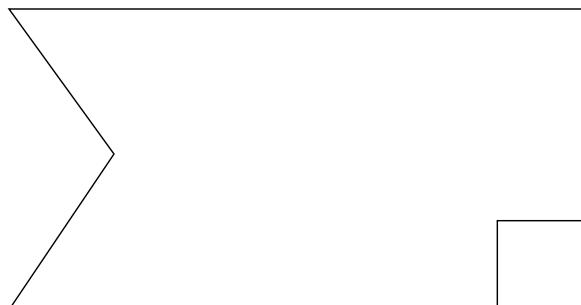
#### Description

Specifies whether merged shapes with 45-degree edges must snap to the grid. Calibre generally only operates on merged data and merging is performed automatically. In hierarchical applications, merging is per-cell, not necessarily across the hierarchy.

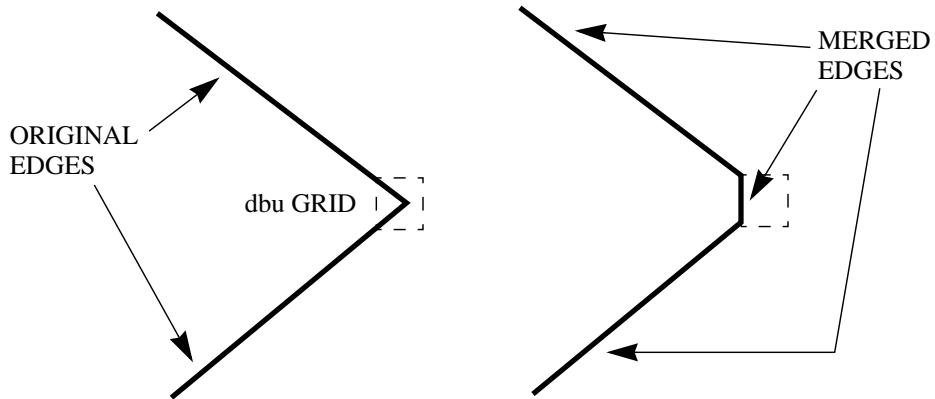
As an example, consider the following two shapes on a drawn, or original layer:



Assume that the intersection point of the two 45-degree edges is not on a database grid. When creating the merged shape below one or both of the 45-degree edges will need to snap, resulting in a merged shape with slightly non-45-degree edges even though the original input was octagonal.



However, if DRC Boolean Nosnap YES is specified, then the merged shape remains octagonal at the expense of inserting a 1 dbu edge as shown below:



See also [Snap Offgrid](#).

## Examples

```
DRC BOOLEAN NOSNAP45 NO
```

## DRC Cell Name

Specification statement

### DRC CELL NAME NO

### DRC CELL NAME YES [CELL SPACE] [XFORM] [ALL]

Used only in Calibre nmDRC-H.

#### Parameters

- **NO**

Keyword that specifies not to append the cell instance name associated with each nmDRC result to its signature line in the nmDRC results database. This is the default behavior when you do not include this statement in the rule file.

- **YES [CELL SPACE] [XFORM] [ALL]**

Keyword set that specifies to append the cell name associated with each nmDRC result to its signature line in the nmDRC results database. When you specify YES, you can also specify any or all of the following optional keywords:

**CELL SPACE** — Specifies to output the coordinates of the nmDRC result in the coordinate space of the cell in which it is instantiated. Top-level coordinate space is the default if you specify YES.

**XFORM** — Specifies to append the nmDRC result's transformation information to its signature line in the nmDRC results database. This keyword is useful if you want to view results in cell-level coordinates in Calibre RVE.

**ALL** — Specifies to list the cell name and transformation information after every nmDRC result in the database. By default the tool does not append this information to results that are identical to previous results.

#### Description

Specifies whether to append the cell name associated with each nmDRC result to its signature line in a hierarchical [DRC Results Database](#) or [DRC Check Map](#) results. This statement applies to ASCII nmDRC results databases only and can be specified at most once.

If you specify YES, then Calibre nmDRC-H places the cell name associated with each nmDRC result on a line beginning with “CN”:

```
TOP 1000
rule_A
15918 15918 1 May 12 08:32:00 2003
DENSITY M1M PGM >= 0.25 WINDOW 100
p 1 4
CN NAND034
-8936900 -4262700
-8935900 -4262700
-8935900 -4261700
-8936900 -4261700
...
...
```

This information, as well as what is described next, enlarges the format of the ASCII nmDRC results database.

By default, Calibre nmDRC-H reports each DRC result in top-level cell coordinates. If you specify YES with the CELL SPACE keyword, the DRC results are reported in the coordinate space of the cells in which they are instantiated. A *c* character is written after the cell name in the signature line. This allows you to determine which coordinate space a DRC result was output in. For example:

```
CN NAND034 c
```

If you specify the XFORM keyword, Calibre nmDRC-H appends the transformation information for each nmDRC result to the end of its signature line in the nmDRC results database. The transformation information is part of a  $3 \times 3$  matrix of six integers that represent the object's transformation from cell space to top-cell space coordinates, if CELL SPACE has also been specified. Without CELL SPACE specified, the matrix is the inverse transformation to cell-space coordinates from top-cell space coordinates.

Cell location coordinates are vectors  $(x, y, 1)$ , where  $x, y$  is a location of a point in database units. Cell location vectors are multiplied by the  $3 \times 3$  transformation matrix to map the location coordinates into the appropriate space.

This is an example for CELL SPACE XFORM:

```
CN NAND034 c 0 1 -1 0 456547 377748
```

It gives nmDRC results where the numbers after the *c* character represent the transformation matrix information.

In this example, the row vectors of the matrix would be  $(\mathbf{0}_{00}, \mathbf{1}_{01}, \mathbf{0}_{02})$ ,  $(-\mathbf{1}_{10}, \mathbf{0}_{11}, \mathbf{0}_{12})$ ,  $(\mathbf{456547}_{20}, \mathbf{377748}_{21}, 1_{22})$ . The bold numbers correspond to the integers after the *c* character. Each cell has a unique set of transformation numbers that represents the transformation of exactly one of its placements in the hierarchy. Generally, this is the lowest, left-most placement of the cell. The transformation numbers are used consistently for all DRC results from that cell.

Matrix elements 00, 01, 10, and 11 control both reflection and rotation, and can have the values -1, 0, or 1. There are eight possible combinations for these four elements, which correspond to the possible unique combinations of reflection and rotation of a cell. Elements 20 and 21 are in database units. Elements 02 and 12 always have values of 0. Element 22 is always 1.

Floating-point transformation numbers are not supported. They are automatically removed from the input layout database by flattening or cloning. Calibre nmDRC-H does not list the cell's transformation data after multiple outputs of the same cell name.

The XFORM keyword is generally used by other tools such as Calibre RVE. Calibre RVE uses the transformation data to allow nmDRC error highlighting in both the cell and top-level contexts. Refer to the section “[Viewing Results in Cell Space](#)” of the *Calibre Interactive and Calibre RVE User’s Manual* for information on how DRC Cell Name affects Calibre RVE behavior.

## DRC Cell Name

---

By default, Calibre nmDRC-H minimizes the ASCII nmDRC results database size by omitting this information for any nmDRC result that is identical to that of the previous nmDRC result (with recurrence). The information is provided for the first nmDRC result in each nmDRC rule check. The ALL keyword overrides this default behavior.

You can specify for Calibre to use cell-level coordinates for results reporting in Calibre Interactive—nmDRC. See “[Reporting DRC Results in Cell Space](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

### Examples

```
DRC CELL NAME YES CELL SPACE XFORM
// output results in cell-space coordinates and provide
// transform matrix elements.
// results are shown in RVE in cell space.
```

## DRC Cell Text

Specification statement

### DRC CELL TEXT {NO | YES}

Used only in Calibre nmDRC-H.

#### Parameters

- **NO**  
Keyword that instructs the tool *not* to use connectivity extraction text at the local cell level. This is the default behavior if you do not include this statement in your rule file.
- **YES**  
Keyword that instructs the tool to use connectivity extraction text at the local cell level.

#### Description

Specifies whether connectivity extraction text is used by the tool at the local cell level of the hierarchy.

The NO keyword instructs nmDRC-H to transform all connectivity text objects that are read in to top-level coordinates. The [Text Depth](#) statement controls the depth from which connectivity text objects are read.

The YES keyword instructs nmDRC-H to behave like nmLVS-H and xRC in relation to connectivity extraction text. That is, connectivity extraction text is used at the local level, in addition to being transformed to the top level. The main application for this capability is the reporting of hierarchical connectivity extraction warnings (short circuit, open circuit, unattached label, and multiply-attached label) by Calibre nmDRC-H. Calibre reports these warnings in cell space rather than the top-level space.

Specifying YES also causes [Net Area Ratio](#) RDB files to show net names for lower-level cells in the NETNAME field. This occurs if a net name is available in the lower-level cell. By default, the NETNAME field in the RDB is empty for lower-level cells.

This statement has no effect on the behavior of [Not] [Net](#) operations.

See also [Text Layer](#).

#### Examples

```
// Do not use connectivity extraction text at the local cell level
// for connectivity reporting purposes.
DRC CELL TEXT NO
```

# DRC Check Map

Specification statement

## DRC CHECK MAP *rule\_check*

```
[{GDSII [layer [datatype]] | OASIS [layer [datatype]]} [PROPERTIES] |  
    ASCII | BINARY]  
[filename | “PIPE command”]  
[PREFIX string] [APPEND string]  
[MAXIMUM RESULTS {maxresults | ALL}]  
[MAXIMUM VERTICES {maxvertex | ALL}]  
[TEXTTAG name]  
[USER MERGED | PSEUDO | USER]  
{{{{AREF cell_name width length [minimum_element_count]}} [MAXIMUM PITCH pitch]}  
[SUBSTITUTE x1 y1 ... xn yn] ...} |  
[AUTOREF [prefix]]}
```

Used only in Calibre nmDRC/nmDRC-H.

## Summary

This statement is used to control the database output structure for nmDRC rule checks.

## Parameters

- ***rule\_check***

A required string that specifies a rule check name or group name.

- **{GDSII [*layer* [*datatype*]] | OASIS [*layer* [*datatype*]]} [PROPERTIES] | ASCII | BINARY**

An optional keyword set that specifies the format of the nmDRC results database. Possible choices are:

**GDSII** [*layer* [*datatype*]] — Specifies a GDSII nmDRC results database. This is the default format for DRC Check Map if the [DRC Results Database](#) format specifies anything except OASIS. The optional strings *layer* and *datatype*, which must be non-negative integers, instruct Calibre to assign the ***rule\_check*** results to GDSII *layer* and *datatype*, respectively.

The range of allowed *layer* and *datatype* numbers is [0, 65535]. The default behavior if you do not include a selection from this set is *layer* = 0, and *datatype* = 0.

You can specify GDS or GDS2 in place of GDSII.

**OASIS** [*layer* [*datatype*]] — Specifies an OASIS nmDRC results database. This is the default format for DRC Check Map if the [DRC Results Database](#) format specifies OASIS. The optional strings *layer* and *datatype*, which must be non-negative integers, instruct Calibre to assign the ***rule\_check*** results to OASIS *layer* and *datatype*, respectively.

The range of allowed *layer* and *datatype* numbers is [0, 65535]. The default behavior if you do not include a selection from this set is *layer* = 0, and *datatype* = 0.

**PROPERTIES** — Causes DFM properties to be exported as GDS text objects (when GDS is specified) or OASIS properties (when OASIS is specified). This behavior only applies for hierarchical runs. This keyword may only be specified with the GDSII or OASIS keywords. The AREF and AUTOREF options may not be used with the PROPERTIES keyword. Using PROPERTIES can increase the output file size. Note that vector DFM properties and DFM Property BY NET ONLY properties are not supported.

**ASCII** — Specifies an ASCII nmDRC results database. For details regarding the differences between flat and hierarchical ASCII results reporting, see “[nmDRC Results Data Storage](#)” in the *Calibre Verification User’s Manual*.

**BINARY** — Specifies a binary nmDRC results database.

The limit for coordinate space is 32-bit for a GDSII nmDRC results database. When writing a GDSII-type nmDRC results database, Calibre aborts if a coordinate exceeds the 32-bit space capacity of GDSII.

Calibre applications print a warning for each **rule\_check** that outputs to a GDSII- or OASIS-type nmDRC results database with a default *layer* number of 0. This may occur because a *layer* number is not specified or because no DRC Check Map statement is specified for the rule check. Setting the environment variable **CALIBRE\_EXIT\_LAYER\_DEFAULT** to a non-null value causes Calibre applications to exit immediately after any such warnings are printed.

- *filename*

An optional name of the output nmDRC results database. Calibre sends the output for the specified **rule\_check** to this file.

If you do not specify this parameter, Calibre sends the output to the filename specified in the [DRC Results Database](#) specification statement.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in chapter 2, “Key Concepts.”

- “**PIPE** *command*”

An optional keyword and argument, in quotation marks, which specify to open a UNIX pipe to the specified *command*. The *command* usually consists of a shell command. This option is used instead of a *filename* to send the results database output to the piped *command*. This option may not be specified with ASCII or BINARY.

- **PREFIX** *string*

Optional keyword and parameter that instructs Calibre to prefix all cell names with *string*. This keyword is only valid for GDSII or OASIS format databases. If specified together with AUTOREF *prefix*, the PREFIX *string* argument precedes the AUTOREF *prefix* argument in all cell names.

- APPEND *string*

Optional keyword and parameter that instructs Calibre to append all cell names with *string*. This keyword is only valid for GDSII or OASIS format databases.

- MAXIMUM RESULTS {*maxresults* | ALL}

An optional keyword set that allows you to specify the maximum number of nmDRC results output for the specified **rule\_check**. This setting takes precedence over the global [DRC Maximum Results](#) statement. Possible choices are:

MAXIMUM RESULTS *maxresults* — Specifies the maximum result count for **rule\_check**, where *maxresults* is a non-negative integer. This is the default behavior has *maxresults* defaulting to 1000. When the maximum results are reached for the specified **rule\_check**, Calibre issues a warning and no further results are added to the nmDRC results database for **rule\_check**.

A warning is issued for each **rule\_check** for which the MAXIMUM RESULTS setting is less than ALL. Setting the environment variable CALIBRE\_EXIT\_MAXIMUM\_RESULTS to a non-null value causes Calibre applications to exit immediately after any such warnings are printed.

MAXIMUM RESULTS ALL — Specifies no limit on the result count (subject to the memory capacity of the host platform) for the **rule\_check**. This is the setting you want to use for RET, MDP, and Litho operations. For 32-bit platforms the maximum is  $2^{32}$ . For 64-bit platforms it is  $2^{64}$ .

- MAXIMUM VERTICES {*maxvertex* | ALL}

An optional keyword set that allows you to specify the maximum number of vertices, per output polygon, for the specified **rule\_check**. This statement takes precedence over the global [DRC Maximum Vertex](#) statement. Possible choices are:

MAXIMUM VERTICES *maxvertex* — Specifies the maximum vertex count, per output polygon for the **rule\_check**, where *maxvertex* is an integer  $\geq 4$ . The default maximum vertex count is 4096.

MAXIMUM VERTICES ALL — Specifies the maximum vertex count per polygon is limited only by the host platform. The maximum is  $2^{32}$ .

When the maximum vertex count is exceeded for a given output polygon, Calibre segments the output polygon into polygons with less than or equal to the maximum vertex limit.

---

**Note**



By default, batch nmDRC can read in OASIS polygons having up to  $2^{32}$  vertices, which is also the maximum number it can output. However, you should avoid having input polygons of greater than 1E06 vertices as this can degrade performance.

---

Batch nmDRC can output GDS polygons having up to  $2^{32}$  vertices, but it can only read in GDS polygons with a maximum of 8192 vertices.

---

- **TEXTTAG *name***

An optional keyword that allows creation of a single text object in the top-level cell of a GDSII-type or OASIS-type nmDRC results database. It is useful for labeling output layers. The text is specified by the *name* parameter. Its layer and texttype are, respectively, the layer and datatype of the DRC Check Map statement (or their defaults). Its location coincides with an arbitrary vertex (in the flat viewpoint) of an arbitrary geometry on the output layer; if none is available, its location is the center of the top cell's extent. This keyword may only be specified if the DRC Check Map type is GDSII or OASIS and is not supported in flat Calibre nmDRC.

- **USER MERGED**

An optional keyword that instructs Calibre nmDRC-H to suppress the output of pseudocells in a hierarchical results database. Geometry in pseudocells is transformed up the hierarchy to the first user cell and then merged. This option is used by the tool as the default except in two cases. If ASCII and MAXIMUM RESULTS ALL are specified, then the USER option is used as the default. If GDSII or OASIS is specified when any Fracture or Litho operations are required by the flow, then the PSEUDO option is used as the default. This keyword applies only to Calibre nmDRC-H and may not be specified with PSEUDO or USER.

- **PSEUDO**

An optional keyword that instructs Calibre nmDRC-H to include pseudocells in a hierarchical results database. This keyword only applies to Calibre nmDRC-H and may not be specified with USER MERGED or USER. This keyword is used as the default if GDSII or OASIS are specified and Fracture or Litho operations are required by the flow.

- **USER**

An optional keyword that instructs Calibre nmDRC-H to suppress the output of pseudocells in a hierarchical results database. Geometry in pseudocells is transformed up the hierarchy to the first user cell but is not merged. This option is used by the tool when ASCII and MAXIMUM RESULTS ALL are specified. This keyword applies only to Calibre nmDRC-H and may not be specified with PSEUDO or USER MERGED.

- **AREF *cell\_name width length [minimum\_element\_count] [MAXIMUM PITCH *pitchx1 y1 ... xn yn*]***

An optional keyword set that instructs Calibre to reduce arrayed rectangles into AREF structures. See the [AREFs](#) section under Description for detailed information.

The *cell\_name*, *width*, *length*, *minimum\_element\_count* (if specified), and SUBSTITUTE optional keyword (if specified) must follow the AREF keyword in the order shown. The parameters are defined as follows:

*cell\_name* — Specifies the name of the target cell in which the AREF structures are placements. The *cell\_name* may be a singleton string variable.

Calibre nmDRC-H does not allow an AREF cell name to match any *placed* structure names in the input layout databases (or any pseudocell names). If this occurs, a warning is issued and output for that DRC Check Map component is suppressed.

*width* — Positive floating-point number, numerical expression, or variable that specifies the width of rectangles to be converted into an AREF structure.

*length* — Positive floating-point number, numerical expression, or variable that specifies the length of rectangles to be converted into an AREF structure.

*minimum\_element\_count* — Specifies the minimum element count of output AREF structures. Must be a positive integer. If the element count is greater than or equal to *minimum\_element\_count*, then the structure is output as an AREF; however, if you specify 1, then an SREF record is used instead. Default is 16 if not specified.

MAXIMUM PITCH *pitch* — Specifies to suppress creation of arrays that exceed the specified *pitch* spacing value. The *pitch* is a floating-point numeric value in user units. The *pitch* is the spacing measured from left edge-to-left edge or from bottom edge-to-bottom edge of adjacent array elements. If the *pitch* is exceeded in either of these cases, then the array is not generated. The default maximum pitch is  $2^{21}$  dbu.

SUBSTITUTE *x1 y1 ... xn yn* — Specifies a polygon with vertices having floating-point coordinates *x1 y1 ... xn yn* to constitute AREF structures rather than a rectangle, which is the default. The polygon must have at least two coordinate pairs, be simple (it cannot be self-intersecting), and orientable (the interior of the polygon must be consistently to the right or the left when traversing the perimeter). Exactly two pair of coordinates are interpreted as the lower-left and upper-right corners (they must be specified in that order) of a rectangle having sides parallel, respectively, to the coordinate axes. These coordinates may be numeric variables, as specified in a [Variable](#) statement. Negative numbers must be enclosed in parentheses. The coordinates are in user units.

This optional keyword set primarily applies to GDSII output from Calibre nmDRC-H, and can be specified any number of times. OASIS placements are stored in arrays automatically, so the AREF keyword is not needed when output is OASIS.

- AUTOREF [*prefix*]

An optional keyword that provides automatic rectangle compaction in GDSII-type nmDRC results databases. This keyword may not be specified with the AREF or OASIS keywords. When specified without the *prefix* parameter, AUTOREF must be the last parameter in the statement to avoid ambiguity.

The optional *prefix* parameter is any string or string variable used as a prefix for all generated cells instead of the default convention, which is to use the prefix “A”. The PREFIX and APPEND options also apply to cell names output by AUTOREF, including PREFIX and APPEND names specified by a [DRC Results Database](#) statement where DRC Check Map writes to that DRC results database.

It is a compiler error for two or more DRC Check Map statements with different AUTOREF prefixes to output to the same DRC results database, or with one prefix explicitly specified and one taking the default value.

AUTOREF is used only by Calibre nmDRC-H; it is ignored (with a warning) by nmDRC-F.

## Description

Specifies how Calibre nmDRC/nmDRC-H outputs the results for a specified rule check name. With this statement, you can generate as many distinct nmDRC results databases as you wish. Each database can be GDSII, OASIS, ASCII, or BINARY, which means that you can generate databases of all types during a single Calibre nmDRC run.

In addition, you can direct the results of the same rule check to multiple nmDRC results databases or the results of multiple rule checks to the same nmDRC results database. You can specify this statement any number of times. This statement is particularly useful in original mask generation applications.

For the specified ***rule\_check***, DRC Check Map overrides global nmDRC results database attributes (where global refers to attributes assigned to all rule checks) as follows:

- Output the results for ***rule\_check*** to the specified filename. This local specification overrides the nmDRC results database filename specified in the [DRC Results Database](#) specification statement.
- Output the results for ***rule\_check*** to a GDSII, OASIS, ASCII, or BINARY nmDRC results database. This local (per rule check) specification overrides the database type specified or defaulted in the DRC Results Database specification statement.
- Change the maximum results output for ***rule\_check*** from that specified globally by the [DRC Maximum Results](#) specification statement. This is useful for applications that combine nmDRC checking and mask preparation in the same rule file.
- Change the maximum vertices for an output polygon for ***rule\_check*** from that specified globally by the [DRC Maximum Vertex](#) specification statement.

In addition, this statement allows you to specify the following local data, which cannot be specified globally with other nmDRC specification statements in the rule file:

- As desired, output the results for the ***rule\_check*** to specific layers and datatypes.
- As desired, during processing of the ***rule\_check***, generate AREF structures, or array placements, from arrayed rectangles of the specified dimension. AREFs are described in more detail later in this section.

To prevent compilation errors due to ambiguous or conflicting rule check mapping to nmDRC results databases, you must adhere to the following DRC Check Map usage rules:

- You must specify each *filename* with the same nmDRC results database type in every DRC Check Map statement in which it appears. If *filename* is specified in the DRC Results Database specification statement, it too must be the same type. The following examples cause compilation errors:

```
DRC CHECK MAP ruleR GDSII databaseX
DRC CHECK MAP ruleS ASCII databaseX // conflicts with GDSII
                                         // for databaseX

DRC RESULTS DATABASE databaseX GDSII
DRC CHECK MAP ruleR ASCII databaseX // similar conflict
```

- If *filename* is not specified, then the nmDRC results database type must match the DRC Results Database specification statement. The following example causes a compilation error:

```
DRC RESULTS DATABASE databaseX // ASCII is the default
DRC CHECK MAP ruleR           // GDSII or OASIS are the defaults;
                                // the causes a conflict
```

In this example, databaseX is ASCII-type by default, whereas ruleR is output in GDSII, which is the default for DRC Check Map. Because ruleR has no assigned *filename*, Calibre nmDRC/nmDRC-H attempts to write the GDSII results for ruleR to the ASCII databaseX. This is an error. The following examples solve the problem:

```
DRC RESULTS DATABASE databaseX // ASCII is the default
DRC CHECK MAP ruleR databaseY // GDSII is written, but to a
                                // different database
```

or

```
DRC RESULTS DATABASE databaseX GDSII
DRC CHECK MAP ruleR // GDSII by default
```

or

```
DRC RESULTS DATABASE databaseX ASCII
DRC CHECK MAP ruleR ASCII
```

- You cannot specify **rule\_check** to the same *filename* in multiple DRC Check Map specifications. The following examples cause compilation errors:

```
DRC CHECK MAP ruleR ... fileF
DRC CHECK MAP ruleR ... fileF // error, same check going to same
                                // results database
```

```
DRC CHECK MAP ruleR
DRC CHECK MAP ruleR // error, similar problem as before
```

```
DRC RESULTS DATABASE fileF
DRC CHECK MAP ruleR ...
DRC CHECK MAP ruleR ... fileF // error, similar problem as before
```

The second and third examples are errors because the nmDRC results database filename defaults to that specified in the DRC Results Database specification statement.

- If you are specifying MAXIMUM RESULTS for a given **rule\_check**, you must specify the **rule\_check** with exactly the same maximum results value in all DRC Check Map specification statements in which **rule\_check** appears. The following examples cause compilation errors:

```
DRC CHECK MAP ruleR ASCII databaseX MAXIMUM RESULTS 500
DRC CHECK MAP ruleR GDSII databaseY MAXIMUM RESULTS 250 // error
```

```
DRC CHECK MAP ruleR ASCII databaseX MAXIMUM RESULTS 500
DRC CHECK MAP ruleR GDSII databaseY // error, default is 1000
```

- If you are specifying a MAXIMUM VERTICES limit for a given *rule\_check*, you must specify all DRC Check Map statements for the given *rule\_check* with the same maximum vertex limit.
- If writing to the same results database as the DRC Results Database statement, and if you use the APPEND or PREFIX keywords, then the specification of these keywords must be consistent for the global results database. For example, this is allowed:

```
DRC RESULTS DATABASE drc.db GDS APPEND _out
DRC CHECK MAP rule1 GDS drc.db
// OK. Global results database suffix applies.
```

However, this is not allowed and causes a compiler error:

```
DRC RESULTS DATABASE drc.db GDS
DRC CHECK MAP rule1 GDS drc.db APPEND _out
// Bad. Does not match corresponding DRC RESULTS DATABASE output
// settings.
```

- If two DRC Check Map statements write to the same results database that is not the global DRC Results Database, and the DRC Check Map statements use the APPEND or PREFIX keywords, then these keyword specifications must be consistent. For example, this is not allowed:

```
DRC CHECK MAP rule1 GDS drc_out.gds APPEND _out
DRC CHECK MAP rule2 GDS drc_out.gds
// Bad. Does not match corresponding DRC CHECK MAP output settings.
```

The library name in the LIBNAME structure of all results databases is drc.db by default. This can be changed using the [DRC Results Database Libname](#) specification statement.

The [Layout Property Audit](#) statement allows property replacement, property insertion, or property modification of file-level properties in OASIS nmDRC results databases.

## PIPE Output

The PIPE option enables you to send the results database output through a UNIX pipe rather than to a file. For example, gzip results output can be created as follows:

```
DRC CHECK MAP rule.1 GDS "PIPE gzip > rule.1_out.gds.gz"
```

If PIPE output is specified, then Calibre nmDRC forks a child process, has it execute “/bin/sh -c *command*”, and pipes the results database output to the standard input of the child process. The PIPE option applies only to GDSII and OASIS output. Exceptions to these restrictions are treated as runtime errors.

## Creating Mask Output Files

DRC Check Map may be used to output mask layout data. This can be done with either original or derived layer data. For example, you might use DRC Check Map to copy GDSII data on an original layer in one file to a different layer number in another file. You can also take a derived layer (like gate = poly AND diffusion) and output it.

When performing this sort of application, use MAXIMUM RESULTS ALL.

Additional details are found under “[Mask nmDRC Results](#)” in the *Calibre Verification User’s Manual*.

### AREFs

The AREF keyword instructs Calibre nmDRC-H to attempt to create GDSII AREF structures, or OASIS repetitions (that is, array placements of the specified cell name), from rectangles of the given width and length (specified in user units). These structures constitute output of the given nmDRC rule check. This can save considerable space in the output database when large numbers of rectangles (which each require 60 bytes in standard GDSII format) are converted to AREF structures; it is especially valuable in conjunction with output generated from the [Rectangles](#) operation.

OASIS placements are stored in arrays automatically, so there is no additional compression if you specify AREF for OASIS output. For this reason, you should not use the AREF keyword with OASIS output in most cases.

AREF output is specified by the optional AREF parameters as in this BNF description:

```
DRC CHECK MAP rule-name ...
[ <aref-specification> [ ... <aref-specification> ] ]
<aref-specification> -> AREF <cell-name> <width> <length>
[ <minimum-element-count> ] ] [ SUBSTITUTE x1 y1 ... xn yn ]
```

By default, AREFs with less than 16 elements are not created. The optional *minimum\_element\_count* parameter, which must be a positive integer, allows you to override this default. Note that this parameter may be as low as 1, in which case the output is an SREF record instead of an AREF.

The AREF keyword cannot be specified if the nmDRC results database type of the statement is neither GDSII nor OASIS.

For example:

```
output_active {
// Planarize the active layer to create the final active mask.
x = ( EXTENT ) NOT ( SIZE active BY 1 ) // Area to planarize.
y = ( RECTANGLES 2 4 1 ) INSIDE x           // Planarization rectangles.
active OR y                                // Final active layer.
}
...
DRC CHECK MAP output_active 2 AREF rect24 2 4
```

In this example, output polygons from nmDRC rule check `output_active`, which are not  $2 \times 4$  rectangles, are sent directly to the nmDRC results database, as before. Calibre nmDRC-H then attempts to create GDSII AREFs from  $2 \times 4$  output rectangles, according to a complex set of internal heuristics intended to maximize the size and number of AREFs created and to minimize the number of rectangles not in AREF structures.

For all  $2 \times 4$  output rectangles (where the dimension along the x-axis is 2), the system attempts to recognize as many array patterns as possible (where each pattern is as large as possible). For each pattern so recognized, an AREF of cell “rect24” of the appropriate dimension and pitch is

output, instead of the rectangles themselves. The process is then repeated for  $4 \times 2$  output rectangles (that is, the dimension along the x-axis is 4); the only difference is that the output AREFs have a rotation component.

Rectangles that are not referenced in the previous algorithm are output to the nmDRC results database. If any AREF is created, then a GDSII structure record, or OASIS CELL record, may also be output representing cell rect24. This structure contains only a single shape on layer #2, representing the rectangle from (0,0) to (2,4).

Calibre nmDRC-H does not allow an AREF cell name to duplicate any *placed* structure name in the input layout database (or any pseudocell name), since the placed structure also has the possibility of being output. In this case, a warning is issued, and AREF output is suppressed for that DRC Check Map AREF component. Otherwise, the structure or cell record is created, as before, unless a cell of the given name actually exists (unplaced) in the input layout database. In that event, the structure or cell record is not created, but the AREF is generated with the *cell\_name*.

Conversion of rectangles into AREFs is completely unrelated to all other DRC Results Database semantics, including maximum result limit, GDSII multiplexing, and so forth. All AREFs are created inside the output GDSII or OASIS cell structures, where the corresponding rectangles would have been output.

The SUBSTITUTE keyword allows an arbitrary polygon to be substituted for the specified rectangle in the output GDSII structure, or OASIS CELL record. The polygon must have at least two coordinates, and be simple and orientable. A two-point polygon is interpreted as an orthogonal rectangle. The coordinates are in user units.

Although not common, multiple AREF outputs may be specified in the same DRC Check Map specification statement. This allows AREF creation for output rectangles of various sizes from the same nmDRC rule check. The restrictions are:

1. An AREF cell name may not appear twice in the same DRC Check Map specification statement.
2. No two AREFs in the same DRC Check Map specification statement can have their width and length parameters equal, in either order.

Restriction 2 is intended to prevent output ambiguity. These restrictions are checked at rule file compilation time.

An AREF cell name in a DRC Check Map specification statement is allowed to duplicate an AREF cell name in another DRC Check Map specification statement, provided that the output layers are equal, the output datatypes are equal, the widths are equal, the lengths are equal, and, if SUBSTITUTE is specified in one, then it must be specified in the other, and the point lists must be identical (again, this restriction is checked at compilation time). This allows AREFs of the same cell to be created by multiple nmDRC rule checks.

Additional details are provided under “[AREF Output](#),” “[GDSII DRC Results Database Format](#),” and “[OASIS DRC Results Database Format](#)” in the *Calibre Verification User’s Manual*.

## AUTOREF

AUTOREF is an optional keyword that causes rectangle compaction in nmDRC-H.

AUTOREF works as follows:

Let R be any GDSII-type (not OASIS) nmDRC results database from the set of all nmDRC results databases generated by the nmDRC-H flow.

Let D be the set of all DRC Check Map specification statements containing the AUTOREF keyword that write to R.

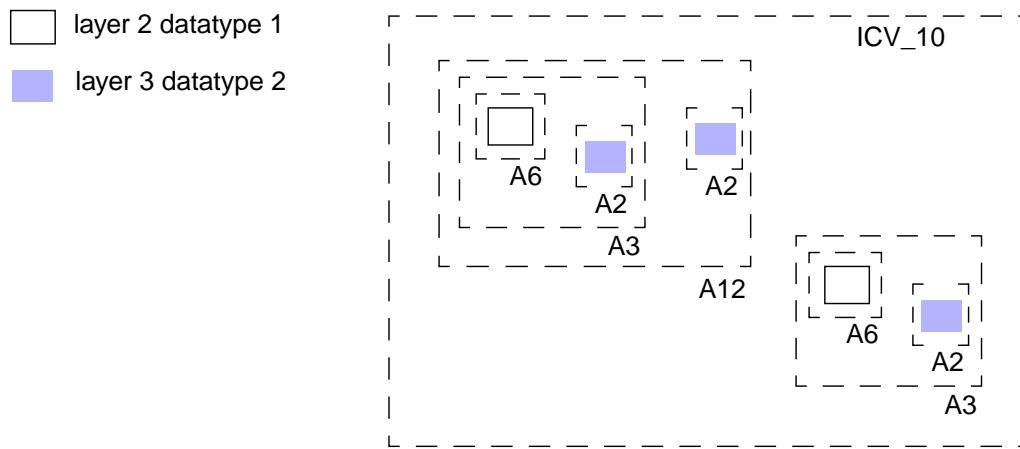
Let C be any Calibre hierarchical database cell that is output to R.

Then the total set of geometric objects G, from the set D, output to C (which transcend both layer number and datatype) consists of two subsets: rectangles O that are orthogonal to the database axes and the remaining polygons P = G - O. The set P is written as usual by the default algorithm. The set O, however, is subject to compaction before being written.

AUTOREF compaction of rectangles is achieved solely by creation of new hierarchical structures (cells, SREFs, and AREFs) in the GDSII results database. Note that these structures are the only available degrees of freedom for compaction allowed by GDSII. The proprietary algorithm is highly efficient (and multi-threaded), and for nmDRC-H results databases with much rectangle content (that is, fill pattern output) significant space reduction may be realized. Hierarchical depth and arrangement is at the discretion of the algorithm.

When a new cell is created, it will be automatically named A<sub>n</sub>, where n is an internally-generated number such that A<sub>n</sub> does not conflict with any existing cell name in the results database. Any given placement of cell A<sub>n</sub> may exist in multiple output hierarchical database cells C — although the algorithm proceeds cell-by-cell, duplicate structures are recognized globally.

To illustrate:



The optional *prefix* parameter replaces the “A” cell name prefix with the string you specify for the *prefix*. The GDSII database size will increase as a function of the length of the *prefix* parameter. A compiler error is issued if you have two or more DRC Check Map statements that output to the same *filename* but have differing AUTOREF *prefix* values, including the default.

The PREFIX and APPEND options apply to cells generated by AUTOREF. The PREFIX *string* will precede the AUTOREF *prefix* if both are specified. This includes when DRC Check Map writes to a DRC results database where the DRC Results Database statement uses the PREFIX keyword.

When specified by itself, the AUTOREF keyword must be the final option in the DRC Check Map statement.

AUTOREF is designed to mitigate the dramatic increase in GDSII database size due to fill synthesis, although the true solution to this problem is migration to OASIS. AUTOREF may also, for example, be used for general compaction of structures like contacts/vias on different layers that are close together or overlap. It may be used to compact output produced by [DFM Transition](#).

## Hierarchy Output

The form of hierarchy output is controlled by the USER MERGED, PSEUDO, and USER keywords. The PSEUDO and USER options exhibit much greater scalability in MT/MTflex runs than the USER MERGED option when you output OASIS results. Details regarding these options are discussed under [Hierarchy Output](#) in the [DRC Results Database](#) section.

The rules for determining whether a given results database file is written with the PSEUDO, USER MERGED, or USER options are as follows:

- If PSEUDO, USER MERGED, or USER is specified locally for the results database file (using DRC Check Map), then the local specification is used.
- Otherwise, the global specification (using DRC Results Database) or its default is used.

A DCM14 compiler error results for any nmDRC results database to have an ambiguous USER MERGED, PSEUDO, or USER specification. For example, this is not allowed:

```
DRC CHECK MAP rule1 GDS 24 out.gds PSEUDO
DRC CHECK MAP rule2 GDS 25 out.gds USER
// bad. out.gds has ambiguous hierarchy output.
```

## File Size Limitations

Limitations on output file size are operating-system dependent. For GDSII, OASIS, and binary output in 32-bit mode, the maximum theoretical output file size is 4 GB. For ASCII output, the practical maximum is 2 GB. If you anticipate output file sizes near these limits, you should use 64-bit mode.

## Examples

### Example 1

Outputting original mask layers:

```
DRC RESULTS DATABASE drc.db GDSII
m1_mask { copy metall }

DRC Check Map m1_mask GDSII 4 m1_mask.gds MAXIMUM RESULTS ALL
// Output metal 1 on layer 4 to m1_mask.gds; show all results
```

**Example 2**

Outputting derived mask layers:

```
DRC RESULTS DATABASE drc.db GDSII

gates { gate = poly and diff
copy gate
}

DRC Check Map gates GDSII gates.gds MAXIMUM RESULTS ALL
// Output gate on layer 0 to gates.gds; show all results
```

**Example 3**

Assume the following rule file excerpt:

```
output_active {
// Planarize the active layer to create the final active mask

x = ( EXTENT ) NOT ( SIZE active BY 1 ) // Area to planarize

y = ( RECTANGLES 2 4 1 ) INSIDE x // Planarization rectangles

active OR y // Final active layer
}
...
DRC CHECK MAP output_active 2 AREF rect24 2 4 8
```

First, Calibre nmDRC-H sends output polygons from the nmDRC rule check `output_active` directly to the nmDRC results database if they are not  $2 \times 4$  rectangles in cell `rect24`. It then attempts to create AREF structures from the remaining  $2 \times 4$  output rectangles if the number of elements is greater than or equal to 8. Calibre nmDRC-H generates AREFs according to a set of internal rules intended to maximize the size and number of generated AREF structures and minimize the number of rectangles not converted to AREF structures.

For all  $2 \times 4$  output rectangles, where the dimension along the x-axis is 2, Calibre nmDRC-H attempts to recognize as many array patterns as possible, where each pattern is as large as possible. For each pattern recognized, an AREF of cell `rect24` of the appropriate dimension and pitch is output, instead of the rectangles themselves. The process is then repeated for  $4 \times 2$  output rectangles, where the dimension along the x-axis is 4. The only difference is that the output AREFs have a rotation component.

Rectangles not converted to AREF structures are output to the nmDRC results database. If Calibre nmDRC-H creates any AREF, then a structure record is also output representing cell `rect24`. This structure contains a single boundary on layer 2 representing the rectangle from (0,0) to (2,4).

# DRC Check Text

Specification statement

**DRC CHECK TEXT** {{COMMENTS | ALL} | RFI} | NONE}

## Parameters

- COMMENTS

Keyword that instructs the tool to output rule check comments (those preceded with an “@” symbol) to the [DRC Results Database](#). These comments appear in Calibre RVE.

COMMENTS RFI is the default if you do not include this statement in the rule file.

- ALL

Keyword that instructs the tool to output the entire text of each rule check to the nmDRC results database. May not be used with COMMENTS.

- RFI

Keyword that instructs the tool to output the rule file title and pathname to the nmDRC results database. COMMENTS RFI is the default if you do not include this statement in the rule file.

- NONE

Keyword that instructs the tool to not output any rule check text or rule file information to the nmDRC results database. May not be used with any other keywords.

## Description

Specifies the amount of rule check text to appear in Calibre nmDRC results databases.

Text consists of @-sign comments, rule file information, and nmDRC rule check text. With this statement you can include all or a subset of the text in the nmDRC results database.

This statement does not apply to GDSII or OASIS nmDRC results databases, and you can only specify it once.

In ICVerify, sets the initial default value for the textmode switch in the \$check\_drc() function.

Virtuoso users can specify DRC Check Text COMMENTS to view the nmDRC check text while stepping through nmDRC or LVS results database errors.

## Examples

```
DRC CHECK TEXT COMMENTS RFI
// place comments, title, and rule file pathname in DRC DB
```

## DRC Exclude False Notch

Specification statement

### DRC EXCLUDE FALSE NOTCH {NO | YES}

Used only in Calibre nmDRC-H applications.

**Note**

 Use of DRC Exclude False Notch YES is strongly discouraged in a production rule file.  
Use the EXCLUDE FALSE optional keyword set of the [External](#) operation instead.

---

### Parameters

- **NO**  
Keyword that instructs Calibre *not* to suppress false notch errors. This is the default behavior if you do not include this statement in your rule file.
- **YES**  
Keyword that instructs Calibre to suppress false notch errors.

### Description

Minimizes the possibility of false errors reported on notches during hierarchical [External](#) operations. A false violation is likely be reported across a bend in a polygon that traverses the hierarchy.

This statement can cause larger runtimes. It is applied to all External operations regardless of whether they produce results or not. It does not apply to flat nmDRC checking and is specified at most once. See “[False Measurement Reduction](#)” in the *Calibre Verification User’s Manual* for more information.

### Examples

```
DRC EXCLUDE FALSE NOTCH NO
```

# DRC Incremental Connect

Specification statement

## DRC INCREMENTAL CONNECT {NO | YES}

Used only in Calibre nmDRC/nmDRC-H.

### Parameters

- **NO**  
Keyword that instructs the tool *not* to enable incremental connectivity. This is the default behavior if you do not include this statement in your rule file.
- **YES**  
Keyword that instructs the tool to enable incremental connectivity.

### Description

Enables incremental connectivity extraction for antenna checking and other specialized connectivity checks. Incremental connectivity causes the tool to connect a subset of the interconnect layers and perform rule checks (antenna checks, for example) based on the connectivity of that subset. See “[Incremental Connectivity and Antenna Checks](#)” in the *Calibre Verification User’s Manual* for detailed information.

In incremental connectivity, the nmDRC application views the rule file as having a partial front-to-back ordering as follows:

```
<layer operations>          //Connectivity zone #0
<connect operations>
<layer operations>          //Connectivity zone #1
                           //((intermediate zone))
<connect operations>
<layer operations>          //Connectivity zone #2
                           //((intermediate zone))
<connect operations>
<layer operations>          //Connectivity zone #N
                           //((final zone))
```

Layer operations include output operations of any nmDRC rule checks. Operations requiring connectivity information in connectivity zone #0 are not allowed. Operations requiring connectivity information in zone #*i*, where *i* > 0, treat the connectivity as if only the [Connect](#) operations prior to connectivity zone #*i* have been executed. Operations requiring connectivity information in zone #*i*, where *i* > 0, are not allowed if that connectivity can only be established by Connect operations after connectivity zone #*i*.

Be aware that you should not place rule checks that do not require connectivity into a connectivity zone as described previously. Such rule checks cause that connectivity zone to be

executed, even when it may not be needed (such as through selective disabling of rule checks that do require connectivity). If this occurs, it is wasteful from a performance standpoint. For this reason, it is a best practice to keep non-connectivity rule checks outside of the incremental connectivity sequence in your rule file.

When setting up the rule file for using connectivity, it is more efficient to establish an exclusive incremental connectivity section and perform only incremental connectivity checks in that section. After that section, you can use that connectivity to do further connectivity-dependent operations that depend upon all connectivity being present. It is wasteful to establish non-incremental connectivity first, break that connectivity (for example, using the [Disconnect](#) operation), and then reestablish incremental connectivity later in the rule file.

Incremental connectivity frequently uses [Net Area Ratio](#) operations. These are particularly useful for antenna checks.

[Not] [Net](#) and [Stamp](#) operations are allowed within any connectivity zone. Use of the [Net](#) operation within an incremental connect sequence allows the checking of certain complex design rules involving shorting of metal nets through diffusion. Net naming (with connect-by-name and potential warnings) occurs within an incremental connect sequence based upon existing connectivity only if a [Not] [Net](#) operation is required in the appropriate connectivity zone. Net naming occurs as usual when all of the [Connect](#) operations have been performed.

The generation of connectivity extraction warnings in an incremental connectivity flow is controlled by the [DRC Incremental Connect Warning](#) statement.

Applications other than Calibre nmDRC and ICrules ignore the DRC Incremental Connect specification statement (at runtime, not compilation time) and treat the rule file as order-independent (and [Connect](#) operations as global). The rule file developer has the responsibility of mitigating connectivity conflicts, if any, in rule files covering multiple applications. This requires greater care by the rule writer with the addition of incremental connectivity. If you perform incremental connectivity checks, the tool optimizes memory usage over the repeated use of the [Copy](#) operation.

**Connectivity Verification Checks for Incremental Connections** — Compile-time connectivity restrictions are enforced for ICverify; however they do not apply when you specify DRC Incremental Connect YES in Calibre nmDRC and nmDRC-H. For example, with DRC Incremental Connect NO (the default), a [Connect](#) layer cannot be derived from an operation requiring pre-established connectivity.

However, for example:

```
X = NET metal1 vdd  
CONNECT X
```

is permissible in a rule file with DRC Incremental Connect YES specified.

One application of this easing of connectivity verification restrictions is that [Net](#) operations may be used to derive subsequent [Connect](#) layers in an incremental connect sequence.

## Restrictions and Recommendations—

- Layer derivations from node-preserving operations (see [page 70](#)) cannot be used across connectivity zones. For example:

```
DRC INCREMENTAL CONNECT YES

connect m1 poly by contact // 1st connectivity zone
x = area m1 > 3
connect m2 m1 by vial      // 2nd connectivity zone
rule {net area x > 10}    // referencing a derived layer from a
                           // node-preserving operation in zone 1
```

is not allowed because Area is a node-preserving operation used in the derivation of layer x. The following is permissible:

```
DRC INCREMENTAL CONNECT YES

connect m1 poly by contact
x = net area m1 > 3
connect m2 m1 by vial
rule {area x > 10}
```

- If DRC Incremental Connect YES is specified and a Net Area Ratio accumulation layer (NARAC) appears in a Connect statement prior to the accumulation layer's use in a later layer operation in the rule file, this causes a NAR16 compiler error. For example:

```
DRC INCREMENTAL CONNECT YES
CONNECT lay1 lay2

a = NET AREA RATIO lay1 >= 0 [AREA(lay1)] ACCUMULATE
b = NET AREA RATIO lay2 >= 0 [AREA(lay2)] ACCUMULATE

CONNECT a b // NARACs appear in CONNECT

a_non_zero = NET AREA RATIO ACCUMULATE a b !=0 [VALUE(a)&&VALUE(b)]
// layer a appears after the CONNECT. Compiler error results.
```

- You cannot use a Mask mode [Label Order](#) operation or a Mask mode [Sconnect](#) operation in the same rule file as the DRC Incremental Connect YES specification statement. Each situation is flagged as a compilation error.
- By default, virtual connection in Calibre nmDRC-H with DRC Incremental Connect YES is only performed within the last Connect block or within a Connect block where there is a [Net](#) or [Not Net](#) operation between the current Connect block and the subsequent one. [Virtual Connect Incremental YES](#) causes virtual connections to be applied in all Connect zones.
- It is recommended that you use incremental connectivity in a multi-threaded environment with hyperscaling (-turbo -hyper command line arguments) for optimal performance. When feasible, it can be advantageous to run incremental connectivity rule checks in a separate run from the rule checks that do not use incremental connectivity.

### Examples

#### Example 1

```
DRC INCREMENTAL CONNECT NO
// do not treat connect statements as order-dependent
```

#### Example 2

```
// Treat the rule file CONNECT statements as order dependent
DRC INCREMENTAL CONNECT YES

// First CONNECT zone
gate = poly AND diff
CONNECT metall1 poly BY co

ant1 {NET AREA RATIO metall1 gate > 100 RDB ant1.db BY LAYER metall1 gate}

// Second CONNECT zone
CONNECT metall2 metall1 BY vial

ant2 {NET AREA RATIO metall2 metall1 OVER gate > 200 RDB ant2.db BY LAYER
metall1 metall2 gate}

// Third CONNECT zone
CONNECT metall3 metall2 BY via2
...
```

# DRC Incremental Connect Warning

Specification statement

## **DRC INCREMENTAL CONNECT WARNING {ENABLE | DISABLE}**

Used only in Calibre nmDRC/nmDRC-H.

### Parameters

- **ENABLE**

Keyword that instructs the tool to generate connectivity extraction warnings during incremental connectivity extraction. This is the default behavior.

- **DISABLE**

Keyword that instructs the tool to disable connectivity extraction warnings during incremental connectivity extraction.

### Description

Controls the generation of connectivity extraction warnings when **DRC Incremental Connect YES** is specified. See “[Incremental Connectivity and Antenna Checks](#)” in the *Calibre Verification User’s Manual* for more information about incremental connectivity applications.

This statement may be specified more than once in your rule file, with each specification controlling the subsequent connectivity zones of the rule file, until the next appearance of this statement in the rule file. This behavior is discussed in detail later in this section.

When incremental connectivity is enabled, text attachment also proceeds incrementally, with the final text attachments occurring after the final block of **Connect** operations has been executed. Although not the only source of connectivity extraction warnings (for example, short and open circuits), text attachment is the primary source. Since text attachment is incomplete prior to conclusion of the final block of Connect statements, connectivity extraction warnings are suppressed until then.

The exception to this behavior is if an intermediate connectivity zone contains a [Not] **Net** operation. Because these operations involve net names, it is normally desirable to see text attachment warnings that may affect the results of these operations; hence, warnings are output after the block of Connect statements that define (that is, directly precede) the zone containing the [Not] Net operations.

There are cases where it is desirable to suppress connectivity extraction warnings in incremental connectivity flows. An example could be an antenna check flow where any warnings (which may be voluminous) could be considered false. Also, for certain reasons, warnings in intermediate zones due to [Not] Net operations may also be considered undesirable.

The generation of connectivity extraction warnings occurs as follows.

## DRC Incremental Connect Warning

---

After the last Connect operation in a block that defines a subsequent connectivity zone is executed, any connectivity extraction warnings are output if either of the following holds:

- This is the last Connect operation in the entire flow.
- This Connect operation begins an intermediate zone that contains a required Net or Not Net operation.

This assumes that the nearest preceding DRC Incremental Connect Warning statement to the current Connect operation is set to ENABLE, either explicitly or by default.

However, the warnings are suppressed if the nearest preceding DRC Incremental Connect Warning statement prior to that block of Connect operations contains the DISABLE keyword.

### Examples

```
DRC INCREMENTAL CONNECT YES
// treat connect statements as order-dependent

DRC INCREMENTAL CONNECT WARNING ENABLE
// enable warnings for the following connectivity zones of the rule file
...
<operations and rule checks>
...
CONNECT ...
CONNECT ...
...
<operations and rule checks with NET and NOT NET>
// connectivity extraction warnings are issued for these operations
...

DRC INCREMENTAL CONNECT WARNING DISABLE
// disable warnings for the following connectivity zones of the rule file

CONNECT ...
CONNECT ...
...
<operations and rule checks with NET and NOT NET>
// connectivity extraction warnings are not issued for these operations
...
```

# DRC Keep Empty

Specification statement

## DRC KEEP EMPTY {YES | NO}

### Parameters

- **YES**

Keyword that instructs the tool to include rule checks containing zero results in the nmDRC results database. This is the default behavior if you do not include this statement in your rule file.

- **NO**

Keyword that instructs the tool *not* to include rule checks containing zero results in the nmDRC results database.

### Description

Specifies whether rule checks containing zero results are sent to the [DRC Results Database](#). This statement can only be specified once.

This statement applies to ASCII, binary, GDSII, and OASIS results database types for nmDRC-H applications. It applies only to ASCII and binary databases for flat applications.

If YES is specified in Calibre nmDRC-H, then all cell and placement records are included in geometric results databases even if the cell, or the placement's cell, respectively, contain no nmDRC results. This is true recursively through the hierarchy. If NO is specified, then empty cell and placement records are not included in geometric results databases.

This statement has no effect upon databases generated using [DRC Check Map](#).

In ICverify, sets the initial default value for the keep empty switch in the \$check\_drc() function.

### Examples

```
// keep empty DRC rule checks
// keep all cell placement records for mask results
DRC KEEP EMPTY YES
```

# DRC Magnify Density

Specification statement

## DRC MAGNIFY DENSITY *value*

Used only in Calibre nmDRC/nmDRC-H.

### Parameters

- *value*

A required, positive, floating-point number or numeric expression, including variables (see [Variable](#)). The *value* is interpreted as a dimensionless magnification factor.

### Description

Specifies the default magnification value for geometric data in RDBs created by [Density](#) operations. The *value* applies if the Density operation specifies an RDB but does not use the MAG keyword. This statement may be specified at most once.

See also [DRC Magnify Results](#) and [Layout Magnify](#).

### Examples

#### Example 1

```
VARIABLE m 2

DRC MAGNIFY DENSITY m // use magnification factor by default
                         // for Density RDB

rule { DENSITY metal < 0.25 [AREA(metal)/AREA())
      WINDOW 50 INSIDE OF LAYER marker
      RDB density_rdb // RDB magnification applies
}
```

# DRC Magnify NAR

Specification statement

## **DRC MAGNIFY NAR** *value*

Used only in Calibre nmDRC/nmDRC-H.

### Parameters

- *value*

A required, positive, floating-point number or numeric expression, including variables (see [Variable](#)). The *value* is interpreted as a dimensionless magnification factor.

### Description

Specifies the default magnification value for geometric data in RDBs created by [Net Area Ratio](#) operations. The *value* applies if the Net Area Ratio operation specifies an RDB but does not use the MAG keyword. The statement may be specified at most once.

See also [DRC Magnify Results](#) and [Layout Magnify](#).

### Examples

#### Example 1

```
VARIABLE m 2

DRC MAGNIFY NAR m // use magnification factor by default
                     // for Net Area Ratio RDB

CONNECT gate

rule { NET AREA RATIO gate > 0 [AREA(gate)]
      RDB NAR_rdb // RDB magnification applies
}
```

# DRC Magnify Results

Specification statement

## DRC MAGNIFY RESULTS *value* [PLACE]

Used only in Calibre nmDRC/nmDRC-H.

### Parameters

- *value*  
A required, positive, floating-point number or numeric expression, including variables (see [Variable](#)).
- PLACE  
Optional keyword that specifies a cell containing the entire output database (except, the actual top-level cell) is created and placed in the top-level cell at the origin, and with the given magnification. This keyword is ignored if the output database format is ASCII.

### Description

Specifies that the Calibre [DRC Results Database](#) is magnified by the given *value* as the database is written. This is especially useful for mask data preparation applications.

The magnification algorithm multiplies all geometric coordinates, text coordinates, placement base points, and array pitches by the given *value* and rounds the numbers back to integers, regardless of an object's hierarchical position.

For mask geometry nmDRC results databases, this algorithm is equivalent, disregarding mathematical round-off error, to placing the entire nmDRC results database at the given magnification into a new top-level cell; the difference is that the new cell is not explicitly created.

During the magnification process, the coordinate values are rounded up or down to the nearest database unit. After magnification, the actual placement of cells may be affected by snapping, causing gaps and jogs. In a flow that includes model-based OPC (Optical and Process Correction), the input layer to OPC must be free of gaps and unintended jogs. The PLACE argument avoids rounding problems that might otherwise introduce gaps and jogs and is especially recommended when using a *value* less than 1.

When using the PLACE argument, multiplication does not occur as the database is being written. Instead, a cell containing the entire output database (except the actual top-level cell) is created and placed in the top-level cell at the origin, and with the given magnification. The name of this cell is:

<top-cell-name>\$M<number>\$

where <number> is the smallest non-negative integer that guarantees uniqueness in the results database of the cell name.

The nmDRC results database magnification value, if specified, is a global nmDRC results database attribute and cannot be locally specified with [DRC Check Map](#) specification statements.

You can specify this statement at most once.

See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

See also [Layout Magnify](#), [Precision](#), [DRC Results Database Precision](#), [DRC Magnify Density](#), and [DRC Magnify NAR](#).

## Examples

Whenever layout database precision is altered, you should specify [Layout Input Exception Severity](#) PRECISION\_RULE\_FILE 1 or 0. Using a rule file Precision that is different from the physical precision introduces an implied scaling of the data, which is compensated for by a [Layout Magnify](#) statement. To avoid possible rounding errors, the product of the Layout Magnify and DRC Magnify Results values should produce an integer, or the magnification value should have a *greater number* of significant figures than the maximum number of significant figures for any coordinate in the design. For all of these examples, “dbu” means database unit.

### Example 1

DRC Magnify results can be used to return data to design scale after doing OPC at wafer scale. Sometimes these conversions can lead to grid snapping because of the ratios involved:

```
// Read in data down to wafer scale
LAYOUT MAGNIFY 0.3
PRECISION 1000
...
// Write data out restored to design scale
DRC MAGNIFY RESULTS 33.33333
DRC RESULTS DATABASE PRECISION 10000
```

The DRC Magnify Results value is set to this:

$\text{output precision} / \text{physical precision} * (1 / \text{layout magnification})$

in this case:

$$10000/1000 * (1/0.3) = 33.33333$$

This is to reduce the amount of grid snapping that occurs. You should truncate the DRC Magnify Results value at a number of significant figures that is the *greater than* the maximum number of significant figures for any coordinate in the design. For example, if the coordinate 550.175 has the maximum number of significant figures in the design, then the DRC Magnify Results value should have at least seven significant figures.

### Example 2

This example provides an alternate method to Example 1 for returning data to design scale. Assume the layout database precision is 1000 dbu/micron. In the GDSII file, 1.5 microns is written as 1500 dbu.

To read in the data down to wafer scale, the Layout Magnify statement specifies the multiplication factor (3) for adjusting the database units, so 1500 dbu becomes 4500 dbu after magnification, or 4.5 microns ( $4500 \text{ dbu} \times 0.001 \text{ micron/dbu}$ ).

The Layout Input Exception Severity PRECISION\_RULE\_FILE 1 statement generates a warning (rather than a fatal error, which is the default) because the rule file precision and the database precision do not match. This statement specifies to use the rule file precision anyway.

The Precision statement sets the ratio of database units to user units, changing the size of 1 dbu from 0.001 microns to 0.0001 microns. The input database size is multiplied by the database precision value ( $4500 \text{ dbu} \times 0.0001 \text{ microns/dbu}$ ) layout so that it equals 0.45 microns, thus achieving a 0.3 magnification of the layout (1.5 microns  $\times 0.3$ ).

```
// Read in data down to wafer scale
LAYOUT MAGNIFY 3
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 1
PRECISION 10000
...
// Write data out restored to design scale
DRC MAGNIFY RESULTS 0.3333333
DRC RESULTS DATABASE PRECISION 10000 /* This is optional in this case */
```

The DRC Magnify Results statement is used to write out the data restored to design scale ( $4500 \text{ dbu} \times 0.3333333 = 1500 \text{ dbu}$ ). The DRC Results Database Precision statement can optionally be used to set the precision value in the results database (it matches the Precision statement value by default).

Alternatively, magnifying by 0.3333333 and changing the DRC Results Database Precision to 1000 would also get back to the original size ( $4500 \text{ dbu} \times 0.3333333 \times 0.001 \text{ micron/dbu} = 1.5 \text{ microns}$ ). However, this method can cause rounding problems due to the reduced precision, which can result in gaps in the data, particularly at cell boundaries. Non-Manhattan geometry can also be snapped in undesirable ways at a reduced precision.

Ensure that the number of significant figures in the magnification statements are enough to guarantee the precision you need in your mask database. The number of significant figures for magnification values should generally be greater than the maximum number of significant figures for any coordinate in the design.

**Example 3**

In the following example, the original database unit size of 0.001 microns is restored to its original scale using the DRC Results Database Precision statement. To keep the scaled size, the output is scaled by 0.72 with the DRC Magnify Results statement.

The PLACE option creates a new cell that contains a single placement of the original top cell and has a magnification of 0.72. Without the PLACE option, the coordinates are magnified by 0.72 and the rounding of the results introduces gaps and jogs.

```
LAYOUT SYSTEM GDS
LAYOUT PATH "original.gds"
LAYOUT PRIMARY "TOP"
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 1
PRECISION 72 100000 // database precision is 1000;
                      // rule file precision is 100000/72

DRC RESULTS DATABASE PRECISION 1000
// want output precision to match input database precision
DRC MAGNIFY RESULTS 0.72 PLACE // maintain the scaled size
                                // 100000/72 * 72/100 = 1000
```

## DRC Map Text

Specification statement

### DRC MAP TEXT {NO | YES}

Used only in Calibre nmDRC-H.

#### Parameters

- **NO**  
Keyword that instructs the tool to output a nmDRC-H results database with no text objects. This is the default behavior if you do not include this statement in your rule file.
- **YES**  
Keyword that instructs the tool to transfer all layout database text objects to the nmDRC-H results database.

#### Description

Specifies whether to transfer text objects in the input layout database to the geometric [DRC Results Database](#). This statement is specified at most once.

You control the depth from which text objects are read with the [DRC Map Text Depth](#) specification statement.

You control the input database layers from which text objects are mapped for output with the [DRC Map Text Layer](#) statement.

The YES keyword is used only when the nmDRC-H results database is GDSII or OASIS. The text objects that are written to the results database have the same hierarchy in both the input and results databases unless a placement was expanded or flattened during hierarchical processing; in that case, the text is moved up the hierarchy as appropriate.

If text objects appear in unwanted locations due to cell expansion by the hierarchical database constructor, you can use the [Layout Preserve Cell List](#) statement together with the [Layout Cell List](#) TEXTED option to prevent expansion of cells containing text objects.

Calibre retains the TEXTTYPE values from input layout databases. In addition, if layer mapping of text occurs with [Layer Map](#) specification statements, then text in the nmDRC results database is on the target layers and the target texttypes (if specified) of any such mapping.

The GDSII layout database presentation attributes for DRC Map Text objects are passed through when YES is specified. These attributes include the optional GDSII PRESENTATION, STRANS, ANGLE, and MAGNIFICATION records (the latter two values are rounded and passed through as integers). (The OASIS format does not support presentation attributes.)

Transformation of these attributes during cell expansion by the hierarchical database constructor is not implemented. That is, justification, rotation, and reflection of text objects as specified in the preserved attributes are not adjusted for transformation if the text objects are brought up the hierarchy due to placement expansion.

Mapping of text from OASIS input to GDSII output is supported.

[Layout Text](#) and [Layout Text File](#) text objects participate with DRC Map Text YES.

## Examples

### Example 1

```
DRC MAP TEXT NO // no text objects written to geometric nmDRC DB
```

### Example 2

This example shows a texttype mapping. By default, the tool uses the texttypes of the input database for output.

```
DRC RESULTS DATABASE cell.gds GDSII
DRC MAXIMUM RESULTS ALL
DRC MAP TEXT YES
DRC MAP TEXT DEPTH ALL
// only map text objects from this layer to output
DRC MAP TEXT LAYER mapped_10_2

LAYER mapped_10_2 1000

LAYER MAP 10 DATATYPE == 2 1000 // layer 10 datatype 2 goes to layer 1000.
LAYER MAP 10 TEXTTYPE == 2 1000 // layer 10 texttype 2 text objects go to
                                // layer 1000 and receive texttype 2 on
                                // output.

mapped_10_2 { COPY mapped_10_2 } // layer 1000 datatype 0 texttype 2
```

### Example 3

This example shows how to prevent cells containing text from being expanded upon output.

```
DRC RESULTS DATABASE top.gds GDSII
DRC MAXIMUM RESULTS ALL
DRC MAP TEXT YES
DRC MAP TEXT DEPTH ALL

// prevent expansion of cells containing text objects
LAYOUT PRESERVE CELL LIST cells
LAYOUT CELL LIST cells "*" TEXTED
```

# DRC Map Text Depth

Specification statement

## DRC MAP TEXT DEPTH {ALL | PRIMARY | *depth*}

Used only in Calibre nmDRC-H.

### Parameters

- **ALL**

Keyword that instructs nmDRC-H to read text objects from all levels of layout hierarchy. This is the default behavior.

- **PRIMARY**

Keyword that instructs nmDRC-H to read text objects from the top-level cell only. This is equivalent to a ***depth*** of 0.

- ***depth***

Non-negative integer, variable, or numeric expression, which instructs nmDRC-H to read text objects down to the hierarchical level of ***depth***. The top level is 0.

### Description

Controls the depth for reading text objects for the [DRC Map Text YES](#) specification statement.

See also [DRC Map Text Layer](#).

### Examples

#### Example 1

Basic syntax.

```
DRC MAP TEXT DEPTH 1
// read text objects from the top cell and one level below
DRC MAP TEXT YES
DRC MAP TEXT LAYER "txt"
DRC RESULTS DATABASE cell.gds GDSII
DRC MAXIMUM RESULTS ALL

TEXT LAYER "txt"
```

#### Example 2

Using an environment variable.

```
VARIABLE level ENVIRONMENT
DRC MAP TEXT DEPTH level

DRC MAP TEXT YES
DRC MAP TEXT LAYER "txt"
DRC RESULTS DATABASE cell.gds GDSII
DRC MAXIMUM RESULTS ALL

TEXT LAYER "txt"
```

# DRC Map Text Layer

Specification statement

```
DRC MAP TEXT LAYER { use_layer [use_layer ...] |  
[use_layer [use_layer ...]] IGNORE ignore_layer [ignore_layer ...] }
```

Used only in Calibre nmDRC-H.

## Parameters

- ***use\_layer***

A name or number of an original layer on which text objects reside in the layout database. Text objects on layers specified using this parameter are mapped to the output. When the IGNORE keyword is not used, this parameter is required. More than one *use\_layer* may be specified.

- **IGNORE *ignore\_layer***

Keyword that instructs the tool to ignore the specified layers when mapping text objects to the output. The *ignore\_layer* is the name or number of an original layer on which text objects reside. More than one *ignore\_layer* may be specified. If no *use\_layer* is specified, then the IGNORE keyword set must be specified.

## Description

Specifies which text objects are output when [DRC Map Text](#) YES is specified. This statement may be specified any number of times.

If a simple layer name is used for the *use\_layer* or *ignore\_layer* parameters, then the semantics are equivalent to the use of its layer number in the statement. If a layer set name is used, then the semantics are equivalent to using all of its constituent simple layer numbers in the statement.

The statement works as follows. Let T be a text object on layer L, for which T is a candidate for output due to specification of DRC Map Text YES. If there are no DRC Map Text Layer specification statements, then T is output. Otherwise T is output if one of the following is true:

- There are no *use\_layer* parameters specified and L is not an *ignore\_layer*.
- There is at least one *use\_layer* specified, L is on a *use\_layer*, and L is not on an *ignore\_layer*.

The *use\_layer* parameter may be used as a *target\_layer* parameter in a Layer Map statement. In such cases, any text objects on the *use\_layer* are then mapped for output according to the Layer Map *target\_layer*.

See also [DRC Map Text Depth](#).

### Examples

#### Example 1

Basic usage:

```
DRC RESULTS DATABASE cell.gds GDSII
DRC MAXIMUM RESULTS ALL
DRC MAP TEXT YES
DRC MAP TEXT DEPTH ALL
DRC MAP TEXT LAYER txt port_txt
// output text objects from these layers only
```

#### Example 2

This example shows the use of text layer mapping. The layout has the following data: polygons on layer 28 datatype 0; text objects on layer 28 texttype 28. The texttype of 28 is maintained for the mapped text objects throughout the flow.

This shows an appropriate method for mapping the layout data to a different layer number in the flow.

```
DRC RESULTS DATABASE cell.gds GDSII
DRC MAXIMUM RESULTS ALL
DRC MAP TEXT YES // send text objects to the output
DRC MAP TEXT DEPTH ALL

LAYER mapped_28 1000

LAYER MAP 28 DATATYPE == 0 1000
LAYER MAP 28 TEXTTYPE == 28 1000 // texttype 28 is maintained

DRC MAP TEXT LAYER 1000 // map only text objects from this layer for
output
```

Notice that this final statement refers to the target layer of the second Layer Map statement. Text objects from the target layer 1000 are mapped for output.

#### Example 3

If you want to output text objects using a different texttype than they intrinsically have, the [Layer Map](#) statement supports this. In the preceding example, the texttype of 28 would be preserved upon output of the text:

```
LAYER MAP 28 TEXTTYPE == 28 1000 // texttype 28 is maintained
```

This statement would change the texttype to 0:

```
LAYER MAP 28 TEXTTYPE == 28 1000 0 // texttype 0 is used
```

# DRC Maximum Cell Name Length

Specification statement

## DRC MAXIMUM CELL NAME LENGTH *value*

Used only in Calibre nmDRC/nmDRC-H.

### Parameters

- *value*

A positive integer indicating the maximum number of characters for cell (and placement) names.

### Description

Specifies the maximum length of cell (and placement) names in a GDSII or OASIS nmDRC results database. Cell names are truncated to the specified *value* prior to being written to the database. The truncation process does not produce duplicate cell names—the truncation length may be decreased for some names to avoid duplication. If truncation is not possible due to avoidance of name duplication, then the cell name is output as is and a warning is issued:

WARNING: DRC MAXIMUM CELL NAME LENGTH truncation of output cell name <cell\_name> to <integer> characters is not possible.

No cell name truncation occurs if this statement is not specified. This statement may be used only once in a rule file.

See also [DRC Results Database](#).

### Examples

```
DRC MAXIMUM CELL NAME LENGTH 8
// cell names only up to first 8 characters go to nmDRC DB
```

# DRC Maximum Results

Specification statement

## DRC MAXIMUM RESULTS {*maxresults* | ALL}

### Parameters

- *maxresults*

Non-negative integer that specifies the maximum result count for an individual rule check in nmDRC execution. When you do not include this statement in your rule file, the default value of number is 1000.

- **ALL**

Keyword that specifies the maximum result count for an individual rule check in nmDRC is limited only by the host platform memory. For 32-bit platforms the maximum is  $2^{32}$ . For 64-bit platforms it is  $2^{64}$ .

### Description

Specifies the nmDRC maximum result count for any given rule check. When the maximum results are generated for a rule check, a warning is issued, and no further results are added to the [DRC Results Database](#) for that rule check. DRC checking then continues for the next rule check. You can specify this statement only once in a rule file.

Calibre applications print a warning if any rule check outputs to a GDSII- or OASIS-type nmDRC results database with a maximum result count less than ALL. The warning is printed for each results database so generated. The intent is to warn of inadvertent use of the default value 1000 when writing mask data versus nmDRC errors. Setting the environment variable CALIBRE\_EXIT\_MAXIMUM\_RESULTS to a non-null value causes Calibre applications to exit immediately after any such warnings are printed.

You can specify the maximum results limit in Calibre Interactive - nmDRC. See “[Limiting the DRC Results Count](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In Pyxis Layout, sets the initial default value for the maxresults switch in the \$check\_drc() function.

See also [DRC Check Map](#).

### Examples

#### Example 1

Limit the result count to 50 results per rule check.

```
DRC MAXIMUM RESULTS 50
```

#### Example 2

Specify no limit to the results count. Use this setting when the output is mask data.

```
DRC MAXIMUM RESULTS ALL
```

# DRC Maximum Unattached Label Warnings

Specification statement

**DRC MAXIMUM UNATTACHED LABEL WARNINGS** *value*

## Parameters

- *value*

Non-negative integer that specifies the maximum number of unattached label warnings issued by the connectivity extraction module.

## Description

Specifies the nmDRC maximum number of warnings of this form:

Label <name> at location <point> [ in cell <name> ] is unattached and has been ignored

If warnings are suppressed by this statement, then a message is output containing the number of suppressed warnings.

## Examples

### Example 1

Limit the number of unattached label warnings to 20.

```
DRC MAXIMUM UNATTACHED LABEL WARNINGS 20
```

## DRC Maximum Vertex

Specification statement

### DRC MAXIMUM VERTEX {*number* | ALL}

Used only in Calibre nmDRC/nmDRC-H.

#### Parameters

- ***number***

Non-negative integer that specifies the maximum vertex count of any nmDRC result polygon that Calibre writes to the nmDRC results database. The value must be an integer greater than or equal to 4. If you do not include this statement in your rule file, the default value of *number* is 4096.

- **ALL**

Keyword that specifies  $2^{32}$  as the maximum vertex count for an output polygon.

#### Description

Specifies the maximum vertex count of any polygon nmDRC result to be written to the [DRC Results Database](#) by Calibre nmDRC/nmDRC-H. This statement can be specified only once in a rule file.

Calibre segments nmDRC result polygons whose vertex count exceeds the specified value and writes the segmented polygons to the nmDRC results database. The segmented polygons have a vertex count less than or equal to that specified (or the default if unspecified).

If polygon segmentation occurs, nmDRC reports a warning once per rule check; the warning indicates the number of polygons segmented like this:

```
WARNING: <number> polygon result(s) segmented in DRC RuleCheck <name>.
```

If your nmDRC Results Database is GDSII format and you either do not specify DRC Maximum Vertex or you specify a number greater than 199, then Calibre issues this warning:

```
WARNING: RuleCheck output to GDSII results database <name> with maximum polygon vertex count of <number> (more than 199).
```

---

#### Note



When you generate GDSII layers using Calibre, be aware that some older downstream mask generating tools may require a 200-vertex limit based on the GDSII specification. In this case, you should use the statement DRC Maximum Vertex 199. Consult your mask vendor regarding this possible limitation.

---

**Note**

 By default, Calibre can read in OASIS polygons having up to  $2^{32}$  vertices, which is also the maximum number it can output. However, you should avoid having input polygons of greater than 1E06 vertices as this can degrade performance.

Calibre can output GDS polygons having up to  $2^{32}$  vertices, but it can only read in GDS polygons with a maximum of 8192 vertices.

---

If your results database output is ASCII, you should not use a maximum vertex count that exceeds 4096. The ASCII database format uses this as an upper limit for vertex counts.

For more information on this topic, see “Polygon Segmentation” in the *Calibre Verification User’s Manual*.

## Examples

```
DRC MAXIMUM VERTEX 4096 //default setting
```

## DRC Print Area

Specification statement

**DRC PRINT AREA** *layer* [*layer* ...]

### Parameters

- *layer*

An original layer or a derived polygon layer. You can specify this parameter any number of times for each statement.

### Description

Computes the flat area for each specified *layer* and prints it to the transcript when the specified layers are generated. A *layer* must actually be read in during the run for the area to be printed. You can specify this statement in the rule file any number of times.

The output from these statements follows the initial OR operation (performed for layer merging) in the transcript. For instance:

```
metal1 = OR metal1
-----
metal1 (...)
CPU TIME ...

DRC PRINT AREA metal1 = <value in user units squared>
```

In hierarchical applications, you can use this statement to provide area data, but there is a performance penalty.

### Examples

```
DRC PRINT AREA poly metal1 metal2
// print the flat areas of these layers
```

# DRC Print Perimeter

Specification statement

**DRC PRINT PERIMETER** *layer* [*layer* ...]

## Parameters

- *layer*

An original layer, or a derived polygon or edge layer. You can specify this parameter any number of times for each statement.

## Description

Computes the flat perimeter for each specified *layer* and prints it to the transcript when the specified layers are generated. A *layer* must actually be read in during the run for the area to be printed. You can specify this statement in the rule file any number of times.

The output from these statements follows the initial OR operation (performed for layer merging) in the transcript. For instance:

```
metal1 = OR metal1
-----
metal1 (...)
CPU TIME ...

DRC PRINT PERIMETER metal1 = <value in user units>
```

In hierarchical applications, you can use this statement to provide perimeter data but there is a performance penalty.

## Examples

```
DRC PRINT PERIMETER poly metal1 metal2
// print the flat perimeters of these layers
```

# DRC Results Database

Specification statement

**DRC RESULTS DATABASE** {*filename* | “**PIPE command**”}  
[ASCII | BINARY |  
{GDSII | GDS | GDS2} |  
OASIS]  
[PREFIX *string*] [APPEND *string*]  
[CBLOCK] [STRICT]  
[USER MERGED | PSEUDO | USER]

Used only in Calibre nmDRC/nmDRC-H.

## Summary

Used to specify the name and type of nmDRC results database.

## Parameters

- ***filename***  
The filename of the nmDRC results database.  
The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.
- **“*PIPE command*”**  
A required keyword and argument if ***filename*** is not used. This keyword set appears in quotation marks and specifies to open a UNIX pipe to the specified ***command***. The ***command*** usually consists of a shell command. This option may not be specified with ASCII or BINARY. If specified with OASIS, then neither CBLOCK nor STRICT may be specified.
- **ASCII**  
Instructs the tool to create a results database in ASCII format. This is the default behavior if you do not specify a keyword from this set.
- **BINARY**  
Instructs the tool to create a results database in a proprietary, binary format.
- **GDSII | GDS | GDS2**  
Instructs the tool to create a results database in GDSII format.
- **OASIS**  
Instructs the tool to create a results database in OASIS format.
- **CBLOCK**  
An optional keyword that instructs Calibre to embed compressed data in an OASIS results database. The number and location of these records are at the discretion of Calibre. In

general, using the CBLOCK parameter provides a larger degree of compression in the output database(s). It may also be faster to write OASIS with CBLOCK records, since less CPU-intensive geometric arraying is done if CBLOCK records are written. This keyword specifies the global behavior for all OASIS databases generated during the run.

- **STRICT**

An optional keyword that instructs Calibre to create strict-mode output for all fields in the table-offsets structure of an OASIS database. This keyword specifies the global behavior for all OASIS databases generated during the run.

- **PREFIX *string***

Optional keyword and parameter that instructs Calibre to prefix all cell names with *string*. This keyword is only valid for GDSII or OASIS format databases. This behavior applies only to results from Calibre nmDRC-H. This keyword set specifies the global default for all GDSII or OASIS databases generated during the run.

- **APPEND *string***

Optional keyword and parameter that instructs Calibre to append all cell names with *string*. This keyword is only valid for GDSII or OASIS format databases. This behavior applies only to results from Calibre nmDRC-H. This keyword set specifies the global default for all GDSII or OASIS databases generated during the run.

- **USER MERGED**

An optional keyword that instructs Calibre nmDRC-H to suppress the output of pseudocells in a hierarchical results database. Geometry in pseudocells is transformed up the hierarchy to the first user cell and then merged. This option is used as the default behavior by the tool except in two cases. If DRC Results Database ... ASCII and [DRC Maximum Results ALL](#) are specified, then the USER option is used as the default. If GDSII or OASIS is specified when any Fracture or Litho operations are required by the flow, then the PSEUDO option is used as the default.

This keyword applies only to Calibre nmDRC-H and may not be specified with PSEUDO or USER.

- **PSEUDO**

An optional keyword that instructs Calibre nmDRC-H to include pseudocells in a hierarchical results database. This keyword only applies to Calibre nmDRC-H and may not be specified with USER MERGED or USER. This keyword is used as the default if GDSII or OASIS are specified and Fracture or Litho operations are required by the flow.

- **USER**

An optional keyword that instructs Calibre nmDRC-H to suppress the output of pseudocells in a hierarchical results database. Geometry in pseudocells is transformed up the hierarchy to the first user cell but is not merged. This option is used as the default by the tool when DRC Results Database ... ASCII and [DRC Maximum Results ALL](#) are specified. This keyword applies only to Calibre nmDRC-H and may not be specified with PSEUDO or USER MERGED.

### Description

Specifies the pathname and type of the Calibre nmDRC results database. This statement must be specified once. Detailed information about Calibre nmDRC results database formats can be found under “[DRC Results Database](#)” in the *Calibre Verification User’s Manual*.

To specify the DRC Results Database in Calibre Interactive—nmDRC, see “[Specifying Outputs for a DRC Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

### ASCII Output

ASCII database format is the default. This format can be read by Calibre RVE for debugging your layout. See “[Using Calibre RVE for DRC](#)” in the *Calibre Interactive and Calibre RVE User’s Manual* for more information.

#### Note



Calibre nmDRC-H only reports a rule check violation *once per cell* for an ASCII results database. The hierarchically lowest, leftmost result is reported. This suppresses multiple instances of the same result, which would normally occur in a flat run. Viewing or debugging of generated layers for a hierarchical run may be easier to do in GDSII or OASIS output because multiple results are not suppressed for these formats.

---

The ASCII DRC Results Database has, as its final entry, a rule check containing the names of all RDB files generated during the run, if any.

The format for this database is described under “[ASCII nmDRC Results Database Format](#)” in the *Calibre Verification User’s Manual*.

### Binary Output

A binary nmDRC results database is approximately two times smaller than its ASCII counterpart. It is primarily used only as an intermediate step to generate database formats from nmDRC results where database size is important. This database is a proprietary Mentor Graphics format.

### GDSII Output

The GDSII format is a standard GDSII stream representation of the nmDRC results database where:

- The GDSII version number in the header record is 6.0.
- The modification and last access times in the BGNLIB and BGNSTR records are the date/time of database creation.
- The library name in the LIBNAME structure is drc.db by default. It can be changed using the [DRC Results Database Libname](#) specification statement.
- The UNITS are drawn from the rule file [Precision](#) and [Unit Length](#) statements, or their defaults.

**Note**

 The limit for coordinate space is 32-bit for a GDSII nmDRC results database. When writing a GDSII-type nmDRC results database, Calibre aborts if a coordinate exceeds the 32-bit space capacity of GDSII.

## OASIS Output

The OASIS format is a standard OASIS representation of the nmDRC results database. It offers advantages over GDSII, particularly in size. OASIS cell placements are stored in arrays automatically. OASIS LAYERNAME records are output to an OASIS results database for all nmDRC rule checks that map to that database. The record consists of the name of the nmDRC rule check and the layer/datatype specified in the appropriate DRC Check Map statement (or the default of layer 0). For information about the Calibre implementation of OASIS, see [Calibre for the Open Artwork System Interchange Standard](#).

The CBLOCK and STRICT parameters are recognized only by 64-bit nmDRC-F and nmDRC-H applications (they are ignored in 32-bit applications). In addition, when explicitly specified, they are global attributes that apply to all OASIS type nmDRC results databases.

Specifying the STRICT parameter instructs Calibre nmDRC to create strict-mode output for all fields in the table-offsets structure. The table-offsets structure contains six pairs of unsigned integers. Each pair consists of a flag field and a corresponding byte-offset field, in the order shown in [Table 4-14](#).

**Table 4-14. Table-offsets Structure**

FLAG	BYTE-OFFSET
cellname-flag	cellname-offset
textstring-flag	textstring-offset
propname-flag	propname-offset
propstring-flag	propstring-offset
layername-flag	layername-offset
xname-flag	xname-offset

The table-offsets structure is stored in the START record if the offset-flag (an unsigned-integer) is 0, and in the END record if the offset-flag is 1. The offset-flag is located in the START record. Calibre always stores the contents of each section referred to in the table-offsets structure after the START record. Each flag is 0 indicating non-strict-mode for the corresponding records and 1 indicating strict-mode.

If STRICT mode is not specified, then all flags and offsets listed in [Table 4-14](#) are zero. If STRICT mode is specified, then CELLNAME, TEXTSTRING, PROPNAME, PROPSTRING, and LAYERNAME records are enabled, the corresponding flags are set to 1, and the byte-offsets are set to address the beginning of each table. The actual location of each table in the file is at the discretion of Calibre. In addition, as required by strict-mode, all references to the

corresponding classes of objects are made exclusively by reference-number. XNAME records are not written by Calibre and remain in non-strict-mode.

Additionally, in strict-mode, each CELLNAME record in the CELLNAME table has an adjoining S\_CELL\_OFFSET standard property whose value is the byte-offset of the CELL record that references the associated CELLNAME record. If compressed output is specified using the CBLOCK parameter, the byte-offset value in the S\_CELL\_OFFSET property is the byte-offset of the CELL record before decompression.

## PIPE Output

The PIPE option enables you to send the results database output through a UNIX pipe rather than to a file. For example, gzip results output can be created as follows:

```
DRC RESULTS DATABASE "PIPE gzip > drc_results.gdsii.gz" GDS PSEUDO
```

If PIPE output is specified, then Calibre nmDRC forks a child process, has it execute “/bin/sh -c *command*”, and pipes the results database output to the standard input of the child process. The PIPE option applies only to GDSII and OASIS output. OASIS output may not have CBLOCK or STRICT records. Exceptions to these restrictions are treated as runtime errors.

## Hierarchy Output

The USER MERGED, PSEUDO, and USER options control the type of results output from hierarchical Calibre applications.

If you specify USER MERGED or USER, then the pseudocells created by nmDRC-H applications are not included in the results database. If you specify USER MERGED, or if it applies as the default, then the data residing within pseudocells is transformed up to the next level of user hierarchy and merged (except for clustered edges). If you specify USER, or if it applies as the default, then the data residing within pseudocells is transformed up to the next level of user hierarchy but is not merged. The main advantage of the USER option over USER MERGED is that USER is typically faster.

For ASCII and BINARY databases, geometry transformation to the top level proceeds as usual. For GDSII or OASIS nmDRC results databases, neither cell records nor placement records for pseudocells are written.

If you specify PSEUDO, Calibre applications include the pseudocells that are internally generated and the geometry is not transformed up the hierarchy. The cells are named ICV\_*n*, where *n* is a unique integer. For an ASCII or binary nmDRC results database, using the PSEUDO keyword can act, in many cases, as an error-suppression mechanism. For layout databases used for mask preparation, using the PSEUDO keyword can dramatically reduce output database size as well as improve performance. PSEUDO is recommended for these situations.

The PSEUDO and USER options exhibit much greater scalability in MT/MTflex runs than does USER MERGED when you specify OASIS results.

The rules for determining whether a given results database file is written with the PSEUDO, USER MERGED, or USER options are as follows:

- If PSEUDO, USER MERGED, or USER is specified locally for the results database file (using [DRC Check Map](#)), then the local specification is used.
- Otherwise, the global specification (using DRC Results Database) is used.

## Multiple Output Databases

You can output multiple nmDRC results databases during one run by using the [DRC Check Map](#) specification statement. Results output by DRC Check Map statements are not sent to the results database specified in a DRC Results Database statement if the database names are not identical in both statements. For example:

```
DRC RESULTS DATABASE "drc.db" GDSII
DRC CHECK MAP rule_x other.db
```

The results sent to the other.db are not sent to drc.db by default. If you want the results to go to both databases, include the following:

```
DRC CHECK MAP rule_x drc.db
```

Calibre nmDRC-H creates a geometric results database with exactly one cell record. The name of the cell is the value of the [Layout Primary](#) statement with the specified *string* appended or prefixed if the APPEND or PREFIX secondary keywords are included. If there is no Layout Primary statement (that is, the input layout system was not GDSII or OASIS), the cell name is drc.

If a DRC Results Database statement uses the APPEND or PREFIX keywords, or both, then these keywords act as global defaults for all GDSII or OASIS results databases generated during the run. A DRC Check Map statement may not have different APPEND or PREFIX keyword settings from the DRC Results Database statement if both statements write to the same file. A DRC Check Map statement may have different APPEND or PREFIX keywords from the global default if that statement writes to a different file than the DRC Results Database statement.

Calibre applications print a warning for each rule check that outputs to a GDSII- or OASIS-type results database with a default layer number of 0. This may occur because a layer number is not specified in the [DRC Check Map](#) specification for the rule check, or because no DRC Check Map statement is specified for the nmDRC rule check. Setting the environment variable `CALIBRE_EXIT_LAYER_DEFAULT` to a non-null value causes Calibre applications to exit immediately after any such warnings are printed.

## File Size Limitations

Limitations on output file size are operating-system dependent. For GDSII, OASIS, and binary output in 32-bit mode, the maximum theoretical output file size is 4 GB. For ASCII output, the practical maximum is 2 GB. If you anticipate output file sizes near these limits, you should use 64-bit mode.

## ICrules Details

Flat ASCII DRC results databases can be converted by ICrules using the RESTORE DRC RESULTS command. You can use the DATABASE EXPORT command in ICrules to create an ASCII results database.

Related statements: [DRC Results Database Precision](#), [DRC Maximum Results](#), [DRC Maximum Cell Name Length](#), [DRC Maximum Vertex](#), [DRC Select Check](#), [DRC Unselect Check](#), [DRC Summary Report](#), and [Layout Property Audit](#).

## Examples

### Example 1

The following statement outputs an ASCII format of the results database to the specified location. You can use Calibre RVE to analyze the results or bring the (flat) results database into ICrules for viewing.

```
DRC RESULTS DATABASE "~/design1/drc_database" ASCII
```

### Example 2

The following statements show how use of the APPEND keyword affects the resulting cell names. Calibre generates subcells, provided that the data exists in the subcells and the subcells do not need to be flattened. Assume that the original cell names in the database are: TOPCELL, SUBCELLA, and SUBCELLB.

The first statement does not use the APPEND keyword:

```
DRC RESULTS DATABASE drc.db GDSII
```

Calibre does not alter the original cell names in the database in this case. The second statement uses *\_new* as the APPEND string:

```
DRC RESULTS DATABASE drc.db GDSII APPEND _new
```

Calibre nmDRC-H alters the original cell names, and saves them to the nmDRC results database as the following: TOPCELL\_new, SUBCELLA\_new, and SUBCELLB\_new. Flat Calibre nmDRC flattens the input database cells and produces the following file: TOPCELL\_new.

**Example 3**

The following rule file excerpt shows how to output multiple results databases:

```
LAYOUT PATH "in.gds"
LAYOUT PRIMARY "top"

DRC RESULTS DATABASE "out.gds" GDSII APPEND _new

LAYOUT SYSTEM GDSII

LAYER MEx1 1

copy_MEx1 { copy MEx1 }

DRC CHECK MAP copy_MEx1 10 APPEND _new
// results go to layer 10 in out.gds

DRC CHECK MAP copy_MEx1 ASCII drcdb.ascii
// results go to drcdb.ascii
```

## DRC Results Database Libname

Specification statement

### DRC RESULTS DATABASE LIBNAME *name*

Used only in Calibre nmDRC/nmDRC-H.

#### Parameters

- *name*

The value of the [DRC Results Database](#) GDSII LIBNAME record.

#### Description

Specifies the value of the LIBNAME record in GDSII nmDRC results databases produced by Calibre nmDRC applications. The default LIBNAME value is drc.db.

This statement may be specified at most once. The LIBNAME specification is a global nmDRC results database attribute and it applies to all GDSII nmDRC results databases created during a Calibre nmDRC run.

See also [DRC Check Map](#).

#### Examples

```
DRC Results Database Libname "design_mask"
// assigns design_mask as the global LIBNAME to GDSII
// nmDRC results databases
```

# DRC Results Database Precision

Specification statement

**DRC RESULTS DATABASE PRECISION** *number* | {*integer1 integer2*}

Used only in Calibre nmDRC/nmDRC-H.

## Parameters

- ***number***  
A positive, floating-point number or numeric variable that specifies the results database precision.
- ***integer1 integer2***  
Positive integers or numeric variables where *integer2* / *integer1* becomes the database precision.

## Description

Specifies the precision value of the [DRC Results Database](#). It has no effect on coordinate values output to the database. You can specify this statement at most once.

By default, the precision of nmDRC results databases generated by Calibre nmDRC applications is set to the value of the [Precision](#) specification statement (its default is 1000). For a GDSII nmDRC results database, the precision value is the database units / user units ratio specified in the UNITS record of the database. For an OASIS results database, the precision value is an integer or a rational number. For ASCII binary nmDRC results databases, the precision value is an integer or a floating-point value. For binary results databases, the precision value is rounded to the nearest integer.

If A and B are integers, specifying DRC Results Database Precision A B is equivalent to DRC Results Database Precision (B/A). For example, DRC Results Database Precision 10 1000 is equivalent to DRC Results Database Precision 100.

If writing out mask data and magnifying the layout by a value that is less than 1 (such as for [DRC Magnify Results](#)), it is recommended that the output database precision be greater than the input physical precision by a factor of 10. This reduces the number of gaps in layers due to grid snapping.

This statement controls the precision of [Density](#) and [Net Area Ratio](#) ASCII results databases (RDBs).

The DRC Results Database Precision value is a global nmDRC results database attribute and cannot be locally specified with [DRC Check Map](#) specification statements.

See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

See also [DRC Magnify Results](#).

## Examples

The following example shows how to increase DRC measurement precision by a factor of 10 and then write out the results using the same precision as the layout.

```
// database precision is 0.001
// increase measurement precision by factor of 10
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 1
PRECISION 10000
LAYOUT MAGNIFY 10
...
// write the results at the layout values
DRC MAGNIFY RESULTS 0.1
DRC RESULTS DATABASE PRECISION 1000
```

# DRC Select Check

Specification statement

**DRC SELECT CHECK** *rule\_check* [*rule\_check* ...]

## Parameters

- *rule\_check*

The name of a rule check statement or rule check group from the rule file. You can specify *rule\_check* any number of times in one statement.

## Description

Allows you to specify the nmDRC rule checks to be executed. By default, the tool selects all nmDRC rule checks in the rule file when the rule file is compiled.

This statement modifies this selection process according to the following algorithm:

1. The tool selects all rule checks if there are no DRC Select Check specification statements in the rule file. Otherwise, only those rule checks specified in DRC Select Check statements are selected.
2. The tool unselects all rule checks specified in [DRC Unselect Check](#) specification statements in the rule file, if specified.

If a rule check is specified with [DFM Select Check](#) or [ERC Select Check](#), it will not execute when Calibre is run in -drc mode, unless it is also specified with DRC Select Check.

You can specify which checks to run in Calibre Interactive—nmDRC. See “[Selecting the Checks to Perform in a DRC Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## Implications for Mask Results Output

The rule checks that are active for a given run comprise the *check set*. If you change the check set by using DRC Select Check or similar statements, this can change the layers that are read in from the layout database and the layer derivations that are executed for the run. Layers (both original and derived) that are *not needed* for the check set *are not processed*. This can affect which layers are output, the extents of cells, and the hierarchy of any mask (geometric) results database.

See “[Rule Check Selection and Mask Results Databases](#)” in the *Calibre Verification User’s Manual* for additional information.

See also [Group](#) and [DRC Select Check By Layer](#).

## Examples

```
DRC SELECT CHECK met1.1 met1.2 met1.3
```

Alternatively, you can do this:

```
GROUP met1_checks met1.?
DRC SELECT CHECK met1_checks
```

## DRC Select Check By Layer

Specification statement

**DRC SELECT CHECK BY LAYER** *layer* [*layer* ...]

### Parameters

- *layer*

Required name (not number) of an original layer. Multiple layer names are allowed.

### Description

Specifies the nmDRC rule checks to be executed based upon *original layer names*. DRC Select Check By Layer causes only those rule checks to be run that require the specified layers. By default, the tool selects all nmDRC rule checks in the rule file when the rule file is compiled.

This statement acts in addition to any [DRC Select Check](#) statements in the rule file.

DRC Select Check By Layer works as follows: For each nmDRC rule check, C, in the rule file, let {L1, L2, ... Ln} be the set of all original layers that are required to execute C. That is, the set of all original layers that are:

1. in the derivation tree of all output operations comprising C, or
2. in the derivation tree of connectivity operations (see [Connect](#) and [Sconnect](#)), if connectivity extraction is required to execute C.

If any element of {L1, L2, ... Ln} is a parameter to any DRC Select Check By Layer specification statement, then a [DRC Select Check](#) C specification statement is implicitly generated.

You can specify which checks to run in Calibre Interactive—nmDRC. See “[Selecting the Checks to Perform in a DRC Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

### Implications for Mask Results Output

The rule checks that are active for a given run comprise the *check set*. If you change the check set by using DRC Select Check By Layer or similar statements, this can change the layers that are read in from the layout database and the layer derivations that are executed for the run.

Layers (both original and derived) that are *not needed* for the check set *are not processed*. This can affect which layers are output, the extents of cells, and the hierarchy of any mask (geometric) results database.

See “[Rule Check Selection and Mask Results Databases](#)” in the *Calibre Verification User’s Manual* for additional information.

See also [DRC Unselect Check By Layer](#).

## Examples

```
LAYER met1 1
LAYER met2 2
LAYER met3 3
LAYER via 4

CONNECT met1 met2 BY via
CONNECT met3

// Perform all rule checks involving met1 and met2 dependencies.
DRC SELECT CHECK BY LAYER met1 met2

// This check is selected because met1 is the output layer.
m1.1 { INT met1 < .08 ABUT < 90 SINGULAR }

// This check is selected because met2 is in the derivation tree of x.
m2.4 { x = INT [met2] < .10 OPPOSITE
        length x > 4 }

/* This check is not selected because met3 is not in the DRC SELECT CHECK
BY LAYER statement, and there are no connectivity dependencies with met1
or met2. */
m3.1 { INT met3 < .08 ABUT < 90 SINGULAR}

/* This check is selected because it is connectivity-dependent and the
connectivity of the via layer is derived from layers in the DRC SELECT
CHECK BY LAYER statement. This is true even though the connectivity of via
is not required for this check. */
via2.3 { NOT INTERACT via met3 >= 1 BY NET }
```

# DRC Summary Report

Specification statement

**DRC SUMMARY REPORT** *filename* [REPLACE] [APPEND] [HIER]

## Parameters

- ***filename***

The report file that is either created or appended to during execution.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “Key Concepts”.

- **REPLACE**

Optional keyword that specifies to overwrite the previous contents of the summary file, if any, each time you run nmDRC. This is the default behavior if you do not specify this keyword.

- **APPEND**

Optional keyword that specifies to append to an existing summary report file. This parameter is useful in creating a log of runs in a single file.

- **HIER**

Optional keyword that instructs Calibre nmDRC-H to create a section in the summary report that lists non-empty rule check statistics by layout database cell.

## Description

Specifies the nmDRC summary report filename and method in which it is written. Enables creation of a file containing a summary report of the run, which includes the following information:

- The date and time of the nmDRC run.
- The rule file pathname and title, if specified.
- The top-level layout cell pathname.
- The current working directory and user name.
- A summary of the number of original shapes processed per original layer.
- A summary of the number of results generated per rule check, as well as a listing of which rule checks from the rule file were not performed.
- All warnings generated.
- The total run time, result count, number of original shapes processed, and number of rule checks executed.

If you do not specify this statement, no summary file is generated.

You can specify the DRC Summary Report in Calibre Interactive—nmDRC. See “[Specifying Outputs for a DRC Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In Pyxis Layout, sets the initial filename summary file parameter and default setting for the summarymode switch in the \$check\_drc() function.

## Examples

```
DRC SUMMARY REPORT drc_report APPEND HIER
// write to drc_report by appending it, include hierarchical
// statistics by cell
```

## DRC Tolerance Factor

Specification statement

### DRC TOLERANCE FACTOR *tolerance*

#### Parameters

- *tolerance*

A positive real number or a numeric expression, including variables (see [Variable](#)). User units of length are assumed. The default is 0.

#### Description

Applies to all dimensional check operations ([External](#), [Internal](#), and [Enclosure](#)) whose measurement constraint is of the form  $< a$  as follows:

For any two edges being measured, where either one is skew with respect to the coordinate axes, the measurement constraint value is decreased by the given ***tolerance*** (up to no more than 0 for the tolerance adjustment) prior to the actual measurement.

This statement is primarily intended to suppress false errors on skew geometry, such as 45-degree paths and circular structures where, due to round off error, distances between edges can be slightly less than the process-specified value.

The minimum ***tolerance*** you should use is [1/Precision](#). The rule file Precision represents the number of database units per user unit.

You can specify this statement only once.

#### Examples

```
DRC TOLERANCE FACTOR .003
// decrease constraint values by this amount for measurements involving
// skew edges
```

# DRC Tolerance Factor NAR

Specification statement

## DRC TOLERANCE FACTOR NAR *tolerance*

### Parameters

- *tolerance*

An (undimensioned) floating-point numeric or numeric expression with  $0 \leq \text{tolerance} < 1$ .

### Description

Sets the tolerance value for testing of [Net Area Ratio](#) and [Net Area Ratio Accumulate](#) operation expressions against their constraints. You can specify this statement once. NAR stands for Net Area Ratio.

The following operation:

**NET AREA RATIO *L1 L2 ... constraint [ expression ] ...***

outputs nets whose value (either an area ratio or some other computed value) meets the operation's constraint. Both the value to be tested and the value(s) of the constraint are floating-point numbers. Hence, there must be a certain amount of tolerance in the comparison. This tolerance attempts to avoid insignificant digits, as well as to make flat and hierarchical output (where rounding error differences could be significant) as consistent as possible.

All constraint tests are based upon a definition of equality of floating-point numbers X and Y. This definition considers only absolute, not relative, equality. X and Y are defined as equal if:

$$|X - Y| < \text{epsilon}$$

where epsilon is some small number. For example, X is less than Y if:

- X is not equal to Y, as defined previously, and
- X is less than Y with an exact comparison.

Other constraint values have similar semantics.

Empirical experience sets the default value of epsilon at 1E-4. The *tolerance* of this specification statement becomes epsilon in all equality definitions for constraint tests in all Net Area Ratio operations.

If Net Area Ratio ... ACCUMULATE is not specified and the operation's constraint is of the form  $< 0$ ,  $> 0$ ,  $\leq 0$ ,  $\geq 0$ ,  $\neq 0$ , or  $\neq 0$ , then an exact floating-point test is performed and DRC Tolerance Factor NAR is ignored.

This statement applies only to Net Area Ratio.

**Examples**

```
DRC TOLERANCE FACTOR NAR 0.00025
// increase the Net Area Ratio tolerance for constraint matching to 2.5E-4
...
x = NET AREA RATIO A B > 1
[ SQRT((PERIMETER(A)-PERIMETER(B))^2)/PERIMETER(B) ]
```

# DRC Unselect Check

Specification statement

**DRC UNSELECT CHECK** *rule\_check* [*rule\_check* ...]

## Parameters

- *rule\_check*

A rule check name or group. You can specify this parameter any number of times for each statement.

## Description

Assists in rule check selection for the nmDRC application. By default, the tool selects all nmDRC rule checks for execution when the rule file is compiled. This statement modifies this selection process according to the following sequence:

1. The tool selects all rule checks for execution if there are no [DRC Select Check](#) specification statements in the rule file. Otherwise, only those rule checks specified in DRC Select Check statements are selected.
2. The tool unselects all rule checks specified in DRC Unselect Check specification statements in the rule file, if specified.

This statement can be useful for temporarily skipping a rule check that requires much time or memory to complete. The nmDRC summary report lists unselected checks as “NOT EXECUTED”.

You can specify which checks to run in Calibre Interactive—nmDRC. See “[Selecting the Checks to Perform in a DRC Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## Implications for Mask Results Output

The rule checks that are active for a given run comprise the *check set*. If you change the check set by using DRC Unselect Check or similar statements, this can change the layers that are read in from the layout database and the layer derivations that are executed for the run. Layers (both original and derived) that are *not needed* for the check set *are not processed*. This can affect which layers are output, the extents of cells, and the hierarchy of any mask (geometric) results database.

See “[Rule Check Selection and Mask Results Databases](#)” in the *Calibre Verification User’s Manual* for additional information.

See also [Group](#) and [DRC Unselect Check By Layer](#).

## Examples

```
DRC UNSELECT CHECK contact_1a contact_1b
```

Alternatively, you can do this:

```
GROUP contact_checks contact_?
DRC UNSELECT CHECK contact_checks
```

## DRC Unselect Check By Layer

Specification statement

**DRC UNSELECT CHECK BY LAYER** *layer* [*layer* ...]

### Parameters

- *layer*

Required name (not number) of an original layer. Multiple layer names are allowed.

### Description

DRC Unselect Check By Layer excludes rule checks from the run that require the specified original layers. By default, the tool selects all nmDRC rule checks in the rule file when the rule file is compiled.

This statement acts in addition to any [DRC Unselect Check](#) statements in the rule file.

DRC Unselect Check By Layer works as follows: For each nmDRC rule check, C, in the rule file, let  $\{L_1, L_2, \dots, L_n\}$  be the set of all original layers that are required to execute C. That is, the set of all original layers that satisfy one of the following:

- In the derivation tree of all output operations comprising C.
- In the derivation tree of connectivity operations (see [Connect](#) and [Sconnect](#)), if connectivity extraction is required to execute C.

If any element of  $\{L_1, L_2, \dots, L_n\}$  is a parameter to any DRC Unselect Check By Layer specification statement, then a [DRC Unselect Check](#) C specification statement is implicitly generated.

You can specify which checks to run in Calibre Interactive—nmDRC. See “[Selecting the Checks to Perform in a DRC Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

### Implications for Mask Results Output

The rule checks that are active for a given run comprise the *check set*. If you change the check set by using DRC Unselect Check By Layer or similar statements, this can change the layers that are read in from the layout database and the layer derivations that are executed for the run. Layers (both original and derived) that are *not needed* for the check set *are not processed*. This can affect which layers are output, the extents of cells, and the hierarchy of any mask (geometric) results database.

See “[Rule Check Selection and Mask Results Databases](#)” in the *Calibre Verification User’s Manual* for additional information.

See also [DRC Select Check By Layer](#).

## Examples

```
LAYER met1 1
LAYER met2 2
LAYER met3 3
LAYER via4 4

CONNECT met1 met2 BY via
CONNECT met3

// Do not perform rule checks involving met1 and met2 dependencies.
DRC UNSELECT CHECK BY LAYER met1 met2

// This check is not selected because met1 is the output layer.
m1.1 { INT met1 < .08 ABUT < 90 SINGULAR }

// This check is not selected because met2 is in the derivation tree of x.
m2.4 { x = INT [met2] < .10 OPPOSITE
        length x > 4 }

/* This check is selected because met3 is not in the DRC SELECT CHECK BY
LAYER statement, and there are no connectivity dependencies with met1 or
met2. */
m3.1 { INT met3 < .08 ABUT < 90 SINGULAR}

/* This check is not selected because it is connectivity dependent and the
connectivity of the via layer is derived from layers in the DRC UNSELECT
CHECK BY LAYER statement. This is true even though the connectivity of via
is not required for this check. */
via2.3 { NOT INTERACT via met3 > = 1 BY NET }
```

## Enclose

Layer operation

**ENCLOSE** *layer1* *layer2* [*constraint* [BY NET] [EVEN | ODD]]

### Parameters

- ***layer1***  
An original layer or layer set or a derived polygon layer.
- ***layer2***  
An original layer or layer set or a derived polygon layer.
- ***constraint***  
One of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. This constraint specifies the number of *layer2* polygons a *layer1* polygon must completely enclose to be selected. The constraint must contain non-negative integers.
- **BY NET**  
An optional keyword used with the *constraint* parameter that specifies a *layer1* polygon is selected when the number of distinct nets (not polygons) on *layer2*, enclosed by the *layer1* polygon, meets the specified *constraint*. The connectivity of *layer2* is required and is checked at compilation time.
- **EVEN**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an even number. May not be used with ODD.
- **ODD**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an odd number. May not be used with EVEN.

### Description

Selects all *layer1* polygons that contain any *layer2* polygons as a subset (that is, a logical OR of a *layer1* and *layer2* polygon(s) yields exactly the *layer1* polygon). Edges of *layer1* and *layer2* polygons may be coincident and still satisfy the enclose condition. Can also select all *layer1* polygons that completely enclose a specified number of *layer2* polygons as indicated by a *constraint*. If either input layer is empty, there is no output.

### BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that meet the constraint are output.

## EVEN and ODD Keywords

The EVEN and ODD keywords modify the interpretation of the *constraint*. For example:

**layer1 ENCLOSE layer2 >3 <9 EVEN**

This operation selects polygons from layer1 that enclose four, six, or eight polygons from layer2.

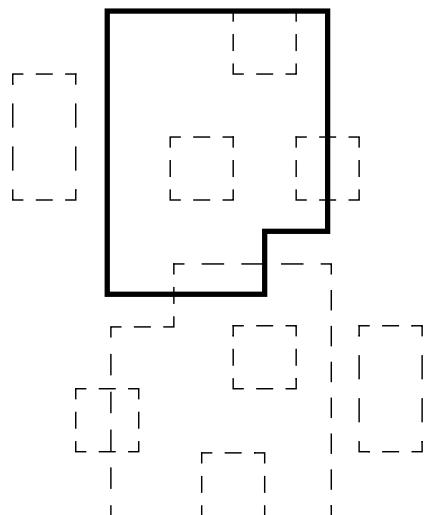
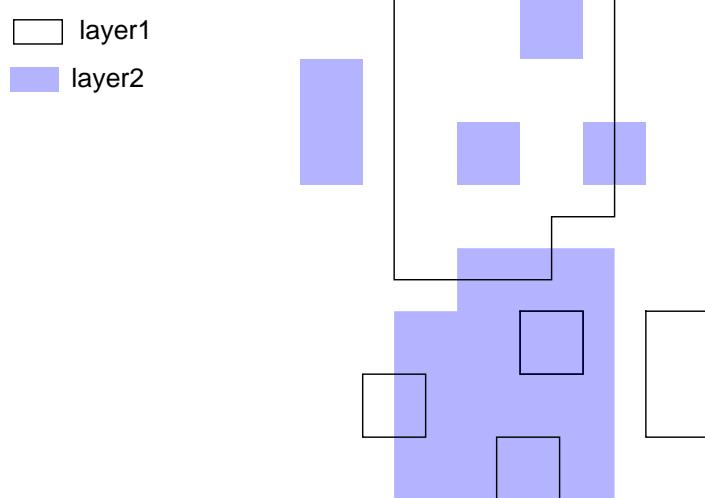
A constraint must be specified with these keywords. It is not useful to specify these keywords if your *constraint* does not allow for the number of polygons to meet the even or odd criterion, such as with == constraints.

See also [Not Enclose](#), [Enclose Rectangle](#), [Enclosure](#), [Rectangle Enclosure](#), [Inside](#), and [Interact](#).

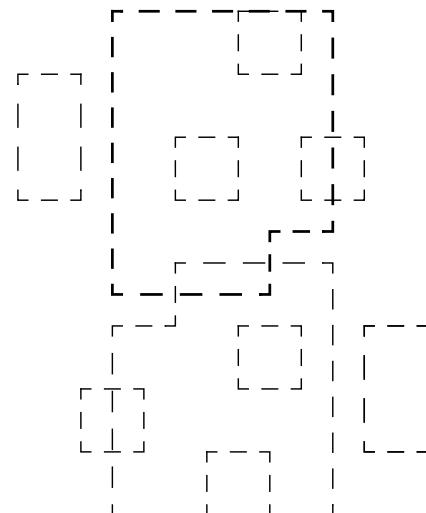
## Examples

Figure 4-75 shows two Enclose operations. The operation on the left selects layer1 polygons that enclose layer2 polygons. The operation on the right selects no layer1 polygons because no polygon meets the constraint.

Figure 4-75. Enclose



enclose layer1 layer2  
//also layer1 enclose layer2



enclose layer1 layer2 > 2

## Enclose Rectangle

Layer operation

**ENCLOSE RECTANGLE *layer width length* [ORTHOGONAL ONLY]**

### Parameters

- ***layer***  
An original or derived polygon layer.
- ***width***  
A positive, floating-point number interpreted in user units.
- ***length***  
A positive, floating-point number interpreted in user units.
- **ORTHOGONAL ONLY**  
An optional keyword which limits selection to polygons that enclose rectangles having sides that are parallel (respectively) to the coordinate axes of the database.

### Description

Selects all polygons on the specified ***layer*** that can enclose a rectangle of the specified ***width*** and ***length***. One or more edges of an enclosed rectangle can be coincident with the enclosing polygon.

The tool may align ***width*** along either the x- or y-axis, but considers only enclosed rectangles having sides parallel (respectively) to the coordinate axes, or rectangles with sides oriented at 45-degree angles to the coordinate axes. To consider only enclosed rectangles having sides parallel to the coordinate axes, specify ORTHOGONAL ONLY.

The operation behaves exactly the same if ***width*** and ***length*** are interchanged. For example, the statement:

```
ENCLOSE RECTANGLE poly 8 2
```

produces the same results as the statement:

```
ENCLOSE RECTANGLE poly 2 8
```

See also [Not Enclose Rectangle](#), [Enclose](#), [Not Enclose](#), [Enclosure](#), [Rectangle Enclosure](#).

### Examples

```
rule { ENCLOSURE RECTANGLE metal 2 4 }
// show all polygons on metal that can enclose a
// 2 x 4 rectangle that is orthogonal to the database axes or
// is oriented at 45 degrees to the database axes.
```

## Enclosure

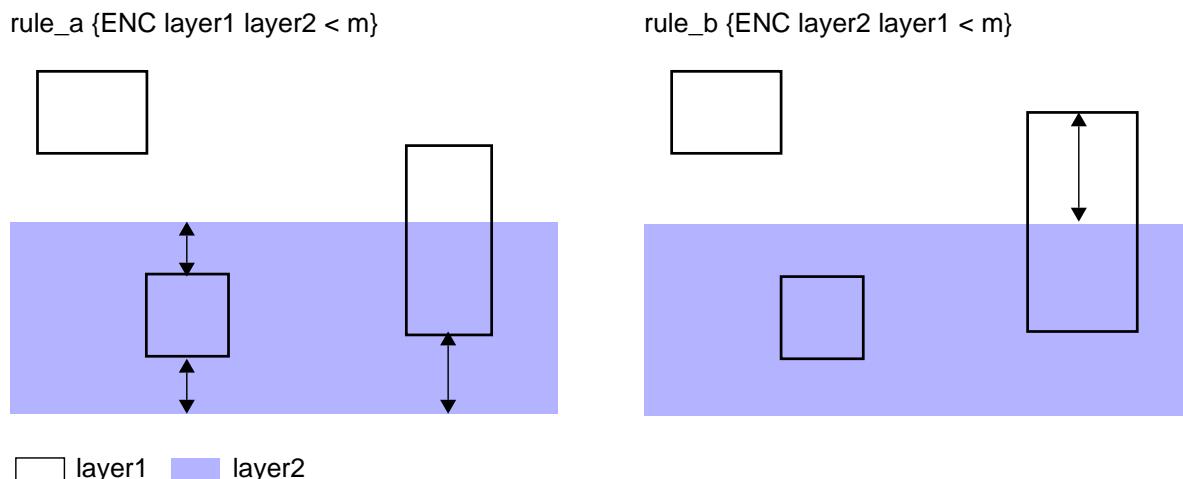
Layer operation

**ENClosure** *layer1 layer2 constraint [metric] [polygon\_containment]*  
*[connectivity\_filter] [orientation\_filter] [projection\_filter] [angled\_filter]*  
*[corner\_filter] [intersection\_filter] [reversal] [EXCLUDE FALSE][region\_output]*

### Summary

Measures the separation between the exterior-facing sides of *layer1* edges and the interior-facing sides of *layer2* edges. The tool outputs measured edge pairs satisfying the given **constraint**. Intersecting edge pairs are not measured by default. There are multiple secondary keyword sets that control the behavior of the ENClosure operation for specialized applications. Principally used for enclosure or extension checks as shown here:

**Figure 4-76. Basic Enclosure Rule Checks**



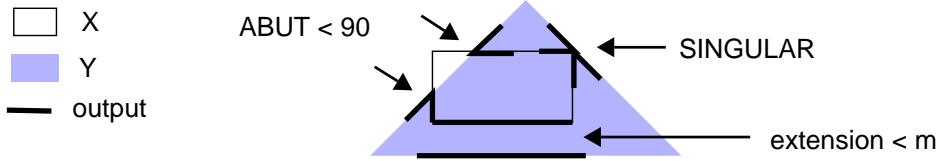
The ENClosure rule checks shown in Figure 4-76 measure the separation between the layer1 and layer2 edges. Notice the order of the layers is important. The line segments with arrows indicate where the measurements occur. By default, edge segments from both layers that meet the measurement constraint are output.

The default ENClosure operation uses the ACUTE ALSO, PARAllel ALSO, NOT PERPendicular, and NOT OBTUSE keywords with the Euclidean measurement metric (which has no keyword).

Figure 4-77 shows the most common type of ENClosure rule check. It uses the ABUT and SINGULAR secondary keywords, which are discussed later in this section under

*intersection\_filter*. ABUT checks intersecting edges and SINGULAR checks single-point interactions. Note that these situations are not checked by default.

**Figure 4-77. Typical Enclosure Check**



```
rule_c {ENC X Y < m ABUT < 90 SINGULAR}
```

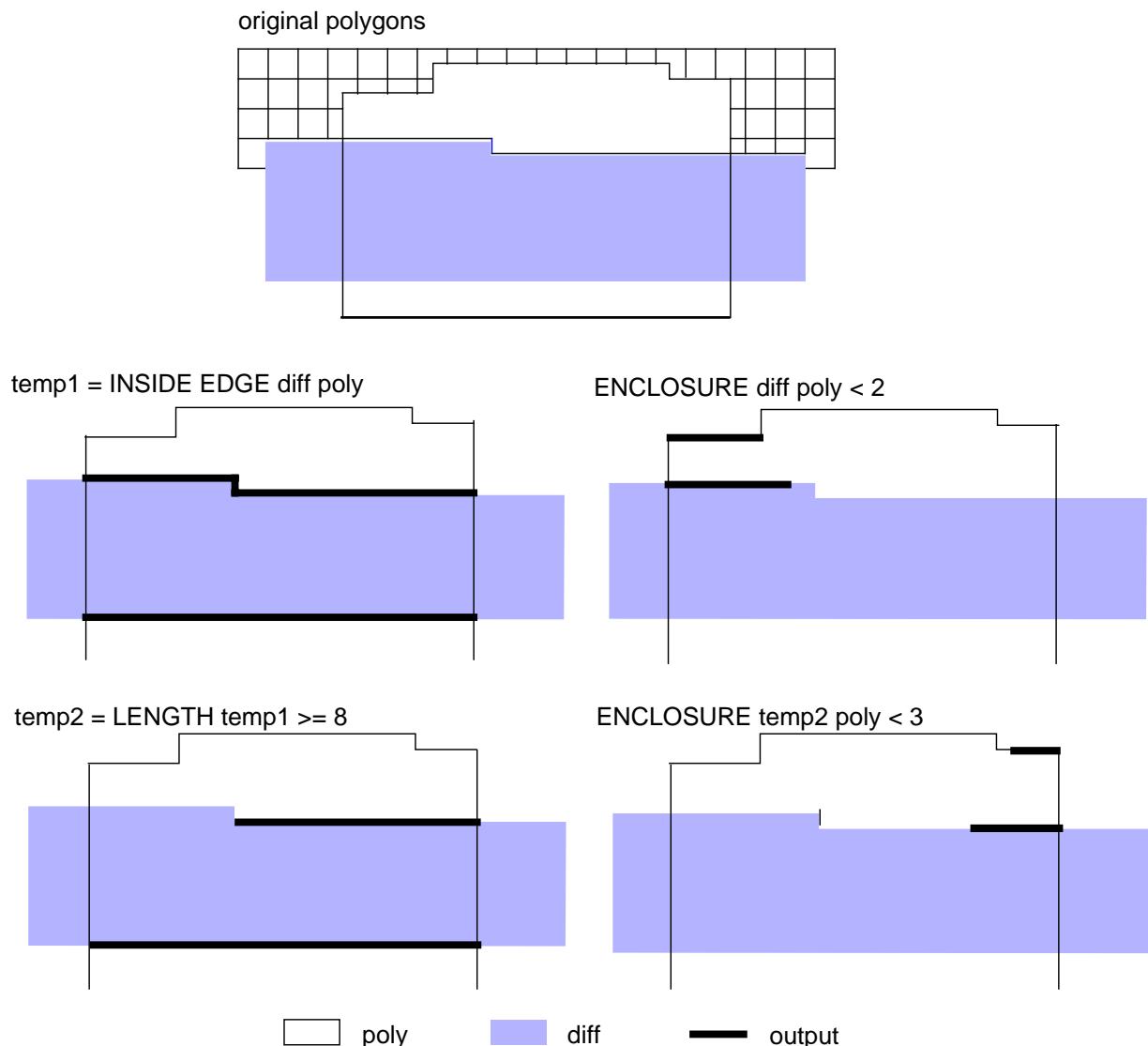
The following rule check shows one use of the ENCclosure operation. Figure 4-78 shows the output of the rule check.

```
// Polysilicon extension over transistor gate must be 2
// microns in general and 3 microns whenever the enclosed
// diffusion edge length is greater than 8 microns.
POLY_EXTENSION {
    temp1 = INSIDE EDGE diff poly
    temp2 = LENGTH temp1 > 8
    ENC diff poly < 2
    ENC temp2 poly < 3
}
```

If ENCclosure is used to generate derived polygon layers (as opposed to edge layers), it is best to use the REGION EXTENTS keyword, like this:

```
x = ENC met1 via1 < 0.04 ABUT < 90 REGION EXTENTS
```

See “[region\\_output](#)” on page 528 for more details about deriving polygon layers with ENCclosure.

**Figure 4-78. Poly Extension Check**

Many fundamental concepts and details of dimensional check operations may be found in the section “Dimensional Check Operations” in the [Calibre Verification User’s Manual](#). You may wish to have this reference open while reading about ENCLOSURE.

For complex multiple-rule enclosure checks, see [Rectangle Enclosure](#). Other related measurement operations include [External](#) and [Internal](#). See also [DRC Tolerance Factor](#), [Not Enclose](#), [Enclose Rectangle](#), and [Not Enclose Rectangle](#).

## Parameters and Description

- ***layer1***

An original layer or layer set, or a derived polygon or edge layer.

Enclosing ***layer1*** in brackets [ ]—called positive edge data—indicates that only edges that meet the ***constraint*** from ***layer1*** should be output. Enclosing ***layer1*** in parenthesis ( )—called negative edge data—indicates that edges from ***layer1*** that do not meet the ***constraint*** be output. Positive and negative edge data may be used to form derived edge layers. See “[Edge-Directed Output](#)” in the *Calibre Verification User’s Manual* for additional details.

- ***layer2***

An original layer or layer set, or a derived polygon or edge layer.

Enclosing ***layer2*** in brackets [ ]—called positive edge data—indicates that only edges that meet the ***constraint*** from ***layer2*** should be output. Enclosing ***layer2*** in parenthesis ( )—called negative edge data— indicates that edges from ***layer2*** that do not meet the ***constraint*** are output. Positive and negative edge data may be used to form derived edge layers. “[Edge-Directed Output](#)” in the *Calibre Verification User’s Manual* for additional details.

Use of positive edge data can be illustrated in this simple example:

```
// check contact enclosure by metall1
enc_cont {
    enc cont [met1] < .1 ABUT < 90 SINGULAR
    // edge-directed output to results database; show only met1 edges
}
```

Positive (or negative) edge data may be used to derive edge layers as illustrated here:

```
// Poly enclosed by diff < .1um may not be > 5 um in length
diff_poly_enc {
    x = enc [poly] diff < .1 //derived edge layer
    length x > 5
}
```

Note that derived error layers, such as the following example (notice the [ ] or ( ) operators are not used, nor is a REGION-type keyword present):

```
x = ENC poly diff < 0.1
```

are usually sent directly to the DRC Results Database and are generally not available for other layer operations to manipulate. However, three DFM operations accept this type of data as input:

- [DFM Analyze](#) — calculates statistics about error data using the COUNT, EC, or EW functions.
- [DFM RDB](#) — writes error layer data to an RDB.
- [DFM Property](#) — filters and/or annotates derived error layers.

- ***constraint***

One of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45, except for  $> a$ ,  $\geq a$ , and  $\neq a$ . Specifies the checking distance, which must have an upper bound. The constraint must contain non-negative real numbers. It is interpreted in user units.

---

**Note**

 ENClosure operations written to test a greater than or equal zero ( $\geq 0$ ) condition within a range constraint (as in  $\geq 0 \leq 4$ ), do not return a touching/coincident ( $=0$ ) condition. The ABUT and SINGULAR secondary keywords should be used to check  $=0$  conditions.

---

- *metric*

Secondary keyword set that instructs dimensional check operations to use a specified edge measurement metric when measuring the separation between edges. By default, the measurement metric is Euclidean. You do not need to specify any keyword to get a Euclidean measurement.

For detailed information about all of these metrics, refer to the “Metrics” section of the *Calibre Verification User’s Manual*.

There are three basic metrics in addition to the default.

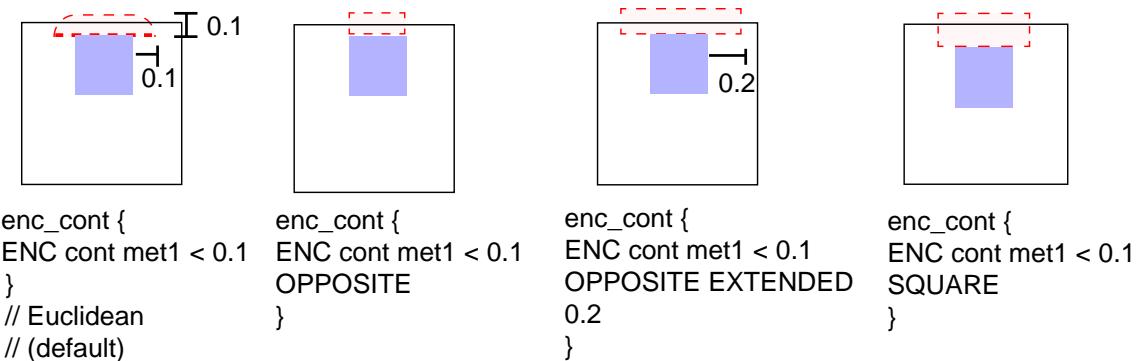
**OPPOSITE** — Specifies extension of the measurement region outward from the edge, but not along the edge, is equal to your **constraint**. It converts the “ $<$ ” constraint to “ $\leq$ ” when measuring intersecting edges. This causes output when intersecting edges abut at 90-degree angles. See [page 523](#) for more discussion of edge intersection.

**OPPOSITE EXTENDED value** — Specifies to use the OPPOSITE metric with an extension of the measurement region along the edge. The *value* is an extension distance in user units and is measured along the edge direction; it must be a positive number.

**SQUARE** — Specifies the extension of the measurement region outward along the edge direction and away from the edge is equal to your **constraint**.

The examples in [Figure 4-79](#) show the four most-commonly-used measurement regions in red. They show only the measurement regions from cont to met1. There are also measurement regions that would be constructed from met1 to cont.

**Figure 4-79. Measurement Regions**



The following are highly specialized metrics used primarily for optical process correction applications. Refer to the “Metrics” section of the *Calibre Verification User’s Manual* for complete details.

**OPPOSITE SYMMETRIC** — Specialized metric based upon the OPPOSITE metric. Used for adjusting edge output for non-orthogonal edges.

**OPPOSITE FSYMMETRIC** — Specialized metric based upon the OPPOSITE SYMMETRIC metric. Uses a special fill-in algorithm that outputs single edges, where disjoint edges might otherwise be output.

**OPPOSITE EXTENDED SYMMETRIC *value*** — Specialized metric based upon the OPPOSITE EXTENDED metric. Used for adjusting edge output for non-orthogonal edges.

**OPPOSITE EXTENDED FSYMMETRIC *value*** — Specialized metric based upon the OPPOSITE EXTENDED SYMMETRIC metric. Uses a special fill-in algorithm that outputs single edges, where disjoint edges might otherwise be output.

**OPPOSITE1** — Unidirectional metric based upon the OPPOSITE metric. Similar to OPPOSITE SYMMETRIC but measures edges in one direction from the first input layer to the second input layer.

**OPPOSITE2** — Unidirectional metric based upon the OPPOSITE metric. Similar to OPPOSITE SYMMETRIC but measures edges in one direction from the second input layer to the first input layer.

**OPPOSITE EXTENDED1 *value*** — Unidirectional metric based upon the OPPOSITE EXTENDED metric. Similar to OPPOSITE EXTENDED SYMMETRIC but measures edges in one direction from the first input layer to the second input layer.

**OPPOSITE EXTENDED2 *value*** — Unidirectional metric based upon the OPPOSITE EXTENDED metric. Similar to OPPOSITE EXTENDED SYMMETRIC but measures edges in one direction from the second input layer to the first input layer.

**SQUARE ORTHOGONAL** — Metric used to simulate mask misalignment in the x- and y-directions.

- *polygon\_containment*

By default, an ENClosure operation between *layer1* and *layer2* (in that order) measures the separation of an exterior-facing edge A from *layer1* and an interior-facing edge B from *layer2* only if A is inside of a polygon having B. Edge B is not inside a polygon from *layer1* and not coincident inside with an edge from *layer1*. The term “edge” here applies equally to segments of edges.

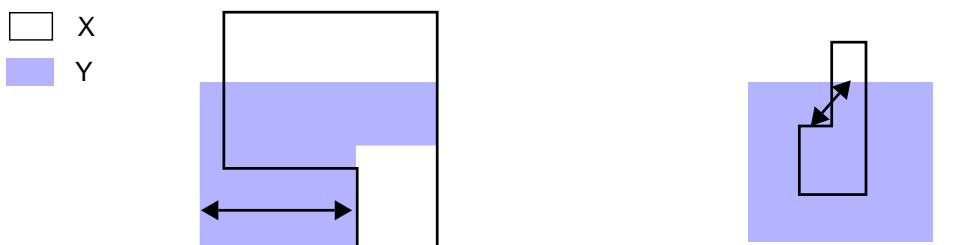
An optional *polygon\_containment* secondary keyword set instructs two-layer dimensional check operations to alter the polygon containment criteria when measuring the separation between edges. Possible choices are:

MEASURE ALL — Specifies to ignore the polygon containment criteria. This allows ENClosure to “see through” polygons that ordinarily it would not.

MEASURE COINcident — Specifies to relax the polygon containment criteria discussed above as follows: edge A may additionally be coincident outside with an edge from layer *layer2* from the same polygon as edge B.

For more about this topic, refer to the “Polygon Containment Criteria” section of the *Calibre Verification User’s Manual*. The example in [Figure 4-80](#) shows how these secondary keyword sets function.

**Figure 4-80. Polygon Containment Criteria Relaxed**



```
rule_c {
  ENC X Y < m
  MEASURE COIN
}
// if m is sufficiently
// large, the segments
// indicated are flagged
```

```
rule_d {
  ENC X Y < m
  MEASURE ALL
}
// if m is sufficiently
// large, the segments
// indicated are flagged
```

- *connectivity\_filter*

A secondary keyword set that instructs dimensional check operations to measure the separation between edges based upon their connectivity. Dimensional check operations ignore connectivity if you do not specify the secondary keywords in this set. Possible choices are:

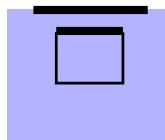
CONNECTED — Specifies to measure only edges from polygons that belong to the same net.

NOT CONNECTED — Specifies to measure only edges from polygons that do *not* belong to the same net.

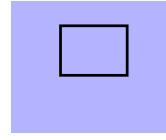
To use these filters, *layer1* and *layer2* must possess valid connectivity. In [Figure 4-81](#), output occurs based upon connectivity information.

**Figure 4-81. Connectivity-Based Checks**

	X
	Y



```
connect X Y
rule_e {
ENC X Y < m
CONNECTED
}
```



```
connect X Y
rule_f {
ENC X Y < m
NOT CONNECTED
}
// no output due to
// connect X Y
```

For related information, see “Node-Preserving Operations” in the [Calibre Verification User’s Manual](#).

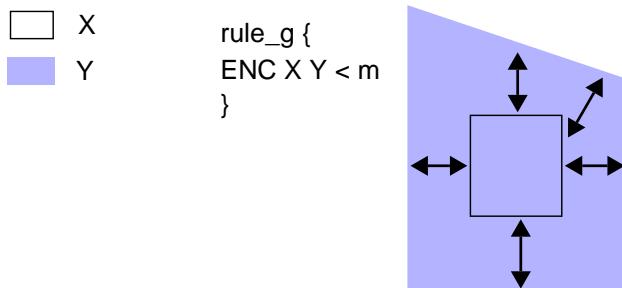
- *orientation\_filter*

A secondary keyword set that instructs the dimensional check operations to measure the separation between edges based upon their appropriate angle or edge orientation. (See the “Appropriateness Criteria” section of the *Calibre Verification User’s Manual*.) You can specify one choice from each of the following four groups in each statement.

The default parameters in this keyword set (PARAllel ALSO, ACUTE ALSO, NOT PERPendicular, NOT OBTUSE) instruct dimensional check operations to measure the separation between the corresponding sides of edges that face each other and that have an appropriate angle of less than 90 degrees.

Figure 4-82 shows the edge orientations that are checked by default using ENCclosure for an appropriate value m. No other orientations are checked unless you select other filters.

**Figure 4-82. Default Edge Orientations Checked by Enclosure**



The secondary keywords having ONLY in them cannot be specified with any other secondary keywords from this set. Possible choices are from the following subsets:

*acute\_filter*

**ACUTE ALSO** — Specifies to measure edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees. This is the default behavior if you do not specify a choice from this subset in the operation.

**ACUTE ONLY** — Specifies to measure only edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same ENCclosure operation.

**NOT ACUTE** — Specifies *not* to measure edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees.

Referring to Figure 4-83:

- ACUTE ALSO checks configurations 1 through 4 (this assumes none of the other default settings are changed).
- ACUTE ONLY checks configurations 3 and 4.
- NOT ACUTE checks configurations 1 and 2.

Orientations 5 and 6 could be checked if the appropriate filters are set (for example, PERPENDICULAR ALSO and OBTUSE ALSO).

**Figure 4-83. Edge Orientations***parallel\_filter*

**PARAlleL ALSO** — Specifies to measure parallel edges in addition to non-parallel edges. This is the default behavior if you do not specify a choice from this subset in the operation.

**PARAlleL ONLY** — Specifies to measure only parallel edges. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same ENClosure operation.

**NOT PARAlleL** — Specifies *not* to measure parallel edges.

Referring to [Figure 4-83](#):

- PARALLEL ALSO checks configurations 1 through 4 (this assumes none of the other default settings are changed).
- PARALLEL ONLY checks configurations 1 and 2.
- NOT PARALLEL checks configurations 3 and 4.

Orientations 5 and 6 could be checked if the appropriate filters are set (for example, PERPENDICULAR ALSO and OBTUSE ALSO).

*perpendicular\_filter*

**NOT PERPendicular** — Specifies *not* to measure perpendicular edges. This is the default behavior if you do not specify a choice from this subset in the operation.

**PERPendicular ONLY** — Specifies to measure only perpendicular edges. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same ENClosure operation.

**PERPendicular ALSO** — Specifies to measure perpendicular edges in addition to other orientations.

Referring to [Figure 4-83](#):

- NOT PERPENDICULAR checks configurations 1 through 4 (this assumes none of the other default settings are changed).
- PERPENDICULAR ONLY checks configuration 5.
- PERPENDICULAR ALSO checks configurations 1 through 5.
- Orientation 6 could be checked if you set OBTUSE ALSO.

*obtuse\_filter*

**NOT OBTUSE** — Specifies *not* to measure edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees. This is the default behavior if you do not specify a choice from this subset in the operation.

**OBTUSE ONLY** — Specifies to measure only edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same dimensional check operation.

**OBTUSE ALSO** — Specifies to measure edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees.

Referring to [Figure 4-83](#):

- NOT OBTUSE checks configurations 1 through 4 (this assumes none of the other default settings are changed).
- OBTUSE ONLY checks configuration 6.
- OBTUSE ALSO checks configurations 1 through 4 and 6.

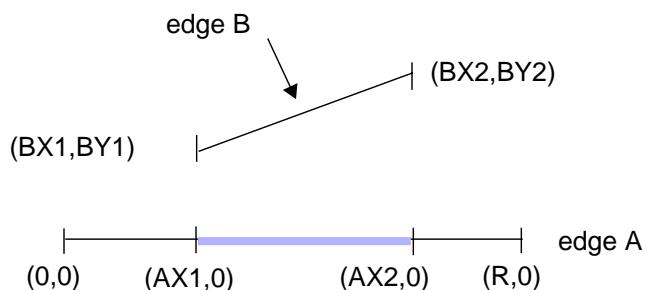
Orientation 5 could be checked if you set PERPENDICULAR ALSO.

- *projection\_filter*

A secondary keyword set that instructs dimensional check operations to measure the separation between edges based upon their mutual edge projection. Note that the projections of edges onto other edges (if any) are not the output, rather the results of the ENClosure operation and associated secondary keywords are output if the specified projection exists.

**Definition of edge projection** — Edge projection is defined as follows (see Figure 4-84): Let A and B be two edges and assume that the endpoint coordinates of A are  $(0,0)$  and  $(R,0)$ ; A and B can always be transformed to achieve this configuration. Let  $BX_1$  and  $BX_2$  be the leftmost and rightmost x-coordinates of edge B, respectively. If  $BX_2 < 0$  or  $BX_1 > R$ , then the projection of B onto A is empty; define the length of the projection of B onto A to be -1 in this case. Otherwise, the projection of B onto A is the edge with endpoints  $(AX_1,0)$  and  $(AX_2,0)$ , where the interval  $AX_1 \leq x \leq AX_2$  is the overlap of the two intervals  $0 \leq w \leq R$  and  $BX_1 \leq z \leq BX_2$ ; the length of this projection is  $AX_2 - AX_1$ . Note that this length may be 0. The length of the projection of A onto B and the length of the projection of B onto A are equal if A and B are parallel; otherwise, they may or may not be equal.

**Figure 4-84. Definition of Edge Projection**



**PROjecting [projection\_length]** — Specifies to measure the separation between two edges only when one edge projects onto the other edge. If *projection\_length* is also specified (it takes the form of a constraint) and the length of any projection conforms to the *projection\_length* constraint then the results are output. The option's default behavior is *projection\_length* set to  $\geq 0$ .

**NOT PROjecting** — Specifies to measure the separation between two edges only when neither edge projects onto the other edge.

**Hierarchical considerations** — For hierarchical applications, projection and projected length may not be accurately determined if the projection occurs at a hierarchical level well outside the design rule distance between the edges. This problem occurs when two non-parallel edges A and B are being measured with PROJECTING or NOT PROJECTING specified, one or both of the edges extends across multiple hierarchical levels, and the actual projecting portions are well outside the design rule distance between the edges. For constrained PROJECTING filters, if you do not specify the PARALLEL ONLY keyword, Calibre issues a warning and sets the PARALLEL ONLY keyword filter. If you have an

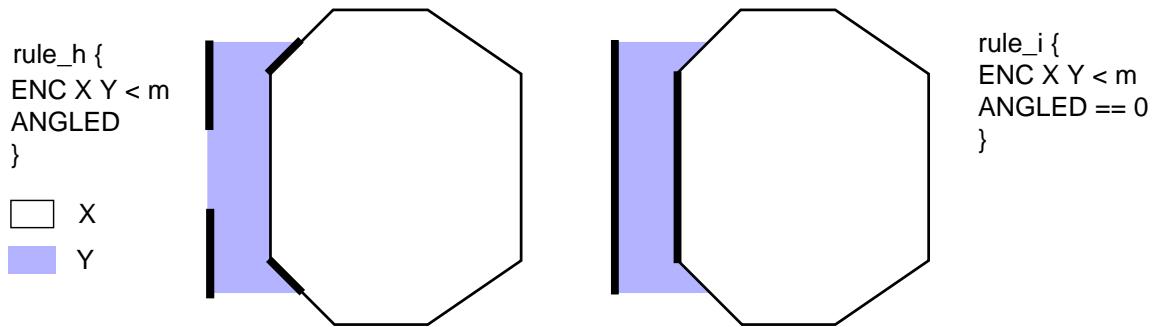
unconstrained PROJECTING filter and you want hierarchical and flat results to be the same, it is highly recommended that PARALLEL ONLY be specified.

- *angled\_filter*

A secondary keyword set that instructs dimensional check operations to measure edge pairs based on orthogonality with respect to the coordinate system axes. The possible choice is:

**ANGLED [angle\_constraint]** — Specifies to measure the two edges only when the number of non-orthogonal edges in the pair meets the given constraint. The constraint is optional and when omitted, defaults to > 0. See Figure 4-85.

**Figure 4-85. ANGLED**



More examples:

```
ENC poly diff < 3 ANGLED == 2
// Measure 2 edges only if both are angled.
```

```
ENC poly diff < 3 ANGLED < 2
// Measure 2 edges only if one or neither is angled.
```

- *corner\_filter*

A secondary keyword set that instructs dimensional check operations to measure edge clusters based on corner-to-corner or corner-to-edge orientation.

You cannot specify this filter with an orientation, projection, or angled filter.

**CORNER TO CORNER** [*corner\_constraint*] — Specifies to measure the edge clusters only if they are in a corner-to-corner configuration. The constraint is optional. When you specify *corner\_constraint*, the only allowed options are == 45 and != 45. The constraint limits the measurement of edges according to the angle that the line segment, which links the opposing corners, makes with the x-axis.

**CORNER TO EDGE** — Specifies to measure the edge clusters only if they are in a corner-to-edge configuration (defined later).

**CORNER** — Specifies to measure the edge clusters only if they are in a corner-to-corner or corner-to-edge configuration.

**NOT CORNER** — Specifies to measure the edge clusters only if they are not in a corner-to-corner or corner-to-edge configuration.

For a two-input dimension check (input layers X and Y), let A be the edge from layer X and B the edge from layer Y. A *convex* 90-degree corner of a polygon is one where the polygon interior covers 90 degrees around the corner point and a *concave* 90-degree corner is one where the polygon interior covers 270 degrees.

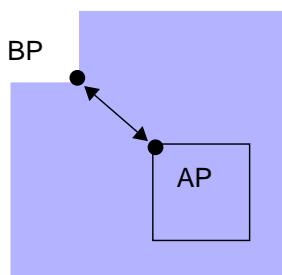
The edge clusters are in a CORNER TO CORNER configuration when:

- The edges are in parallel and do not project.
- Let AP be the point on edge A closest to edge B and let BP be the point on edge B closest to edge A. (These points are guaranteed to be unique due to the parallel and projecting restriction.)
- AP lies at a convex 90-degree corner and BP lies at a concave 90-degree corner. See Figure 4-86.

**Figure 4-86. CORNER TO CORNER**



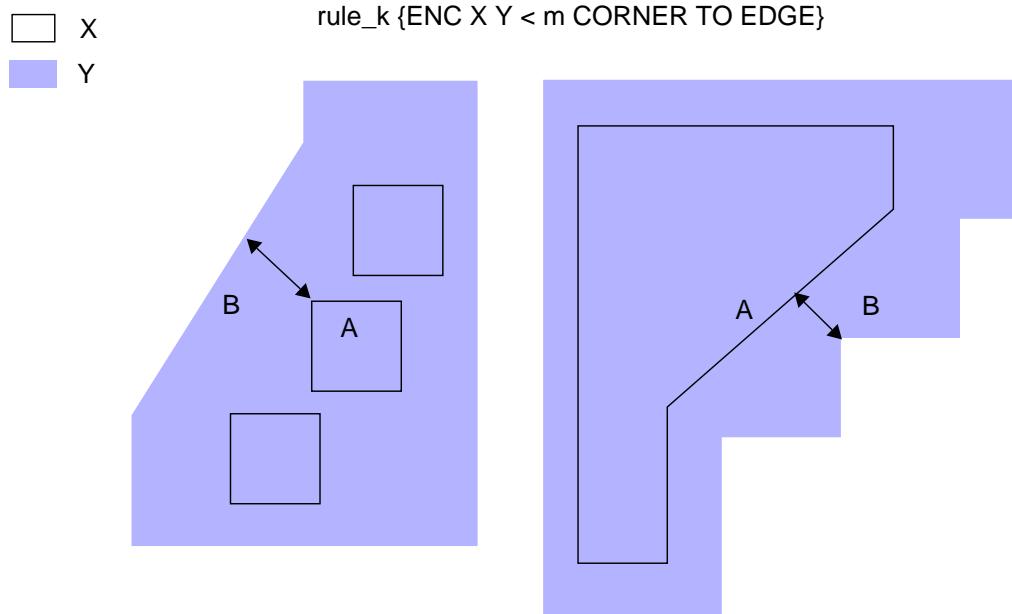
rule\_j {ENC X Y < m CORNER TO CORNER}



The two edges are in a CORNER TO EDGE configuration when these conditions hold:

- o The edges are neither parallel nor perpendicular.
- o Either endpoint of edge A is at a convex 90-degree corner and the corner projects onto the other edge, or either endpoint of edge B is at a concave 90-degree corner and the corner projects onto the other edge (that is, the line bisecting the corner intersects the edge at a unique point and this point can be an endpoint of the edge). See Figure 4-87.

**Figure 4-87. CORNER TO EDGE**



Edges that are not involved in the interactions shown in the previous two examples would satisfy the NOT CORNER condition (assuming they satisfy the ENCclosure condition).

- *intersection\_filter*

An optional parameter that measures edge pairs based upon intersection behaviors. The value of *intersection\_filter* takes the following form:

[ABUT [abut\_constraint]] [OVERLAP] [SINGULAR]  
[{*intersection\_filter* INTERSECTING ONLY}]

You can specify any combination of the secondary keywords ABUT, OVERLAP, and SINGULAR in one operation:

ABUT [*abut\_constraint*] — Specifies that separation between intersecting edges should also be checked. It is highly recommended that you use ABUT < 90 in your rule file unless you have good reasons not to. *Intersecting or abutting edges are not checked by default.* Intersecting edges are measured only if the appropriate angle between them conforms to the optional constraint (interpreted in degrees). With the exception of intersection, the edges must also meet the criteria of the dimensional check operations and any keywords specified from the *connectivity\_filter*. Output from the ABUT condition is in addition to any other output the ENCclosure operation generates.

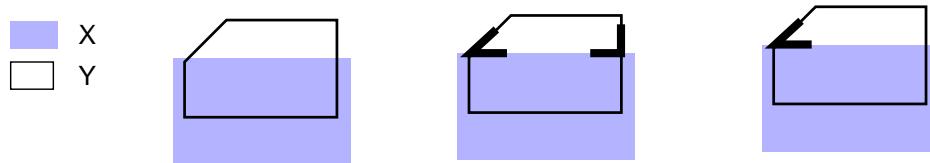
The *abut\_constraint* modifier must contain non-negative real numbers less than 180. Single-operator constraints such as < 90 and > 135 are interpreted as  $\geq 0 < 90$  and  $> 135 < 180$ . With no constraint specified, the default constraint is  $\geq 0 < 180$ .

If the *abut\_constraint* modifier includes zero in its range (for example, < 90, == 0,  $\geq 0 < 45$ ), then any edges A of X and B of Y that are *coincident inside* are also output (because the angle between the exterior side of A and the interior side of B is zero). There is no measurement involved in this event, and polygon containment criteria are not applied.

This measurement of intersecting edges ignores orientation, projection, and corner filters, but does not ignore connectivity, polygon containment, and angled filters.

Abutting edges are not considered by the ENCclosure operation without the ABUT secondary keyword. See Figure 4-88:

**Figure 4-88. ABUT Examples**



rule\_l {  
ENC X Y < m  
// no output}

rule\_m {  
ENC X Y < m  
ABUT}

rule\_n {  
ENC X Y < m  
ABUT < 90}

OVERLAP — Specifies that measurement of the separation between intersecting edges at points where a polygon from one input layer crosses a polygon from the other input layer should also occur. All edges forming the point of overlap are measured as if an unconstrained ABUT parameter was specified (see rule\_m in Figure 4-88). This

overrides any specified ABUT parameter at the point of overlap only. This keyword cannot be used when either of the input layers is a derived edge layer because overlaps cannot be computed in the absence of polygons. Output from the overlap condition is in addition to any other output generated by the ENCLOSURE operation.

The precise definition of a point at which overlap is detected is as follows:

Given a two-layer dimensional check operation between layers A and B, let P be any point where more than one edge from layer A is present and more than one edge from layer B is present (after edge-breaking, see the “Edge Breaking” section of the *Calibre Verification User’s Manual* for details). If at both of the following conditions are met at point P there is overlap:

- An edge from layer A that is outside of, or coincident outside with, a polygon from layer B, and an edge from layer A that is inside of, or coincident inside with, a polygon from layer B.
- An edge from layer B that is outside of, or coincident outside with, a polygon from layer A, and an edge from layer B that is inside of, or coincident inside with, a polygon from layer A.

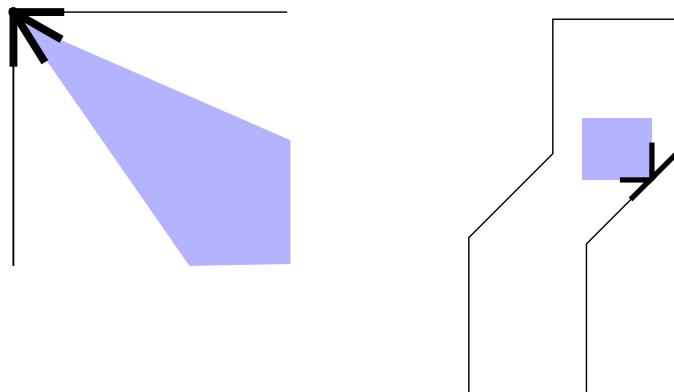
**SINGULAR** — Specifies that the separation between the corresponding sides of intersecting edges at points of polygon singularity should also be measured.

*Singularities* are point-to-edge or point-to-point polygon intersections or self-intersections and are often design rule errors. It is highly recommended that you use SINGULAR in your rule file unless you have good reasons not to because *singularities are not checked by default*. Output from the singular condition is in addition to any other output generated by the ENCLOSURE operation. Figure 4-89 shows examples.

When a dimensional check operation detects a point of singularity, all edges forming the point of singularity are measured as if an unconstrained ABUT parameter were specified in the operation. This overrides any specified ABUT parameter, but only at the point of singularity.

You cannot use the SINGULAR keyword when either of the input layers is a derived edge layer because singularities cannot be computed in the absence of polygons.

**Figure 4-89. SINGULAR Examples**



**INTERSECTING ONLY** — Specifies to measure only between intersecting edges and to ignore non-intersecting edges. Must be preceded with at least one of the other filters in this set. This filter ignores orientation, angled, projection, and corner filters.

Remember, ABUT, OVERLAP, and SINGULAR output their results *in addition to* the typical ENCclosure output. INTERSECTING ONLY limits output to only what the *intersection\_filter* keywords produce. Figures 4-88 and 4-89 show the types of output to expect from INTERSECTING ONLY. Figure 4-76 does not because the measured edges do not intersect.

Here is an example of a common configuration of intersection keywords:

```
// Contact enclosure by poly = 0.6, crossing edges not ok,  
// coincident inside edges not ok, point-to-point touching not  
// ok.  
rule_n {  
ENC contact poly < 0.6 ABUT == 0 OVERLAP SINGULAR  
}
```

-or-

```
rule_o {  
ENC contact poly < 0.6 ABUT  
}
```

Using the second variant makes the code simpler, but assumes rectangular contacts; otherwise, false errors could result in certain configurations.

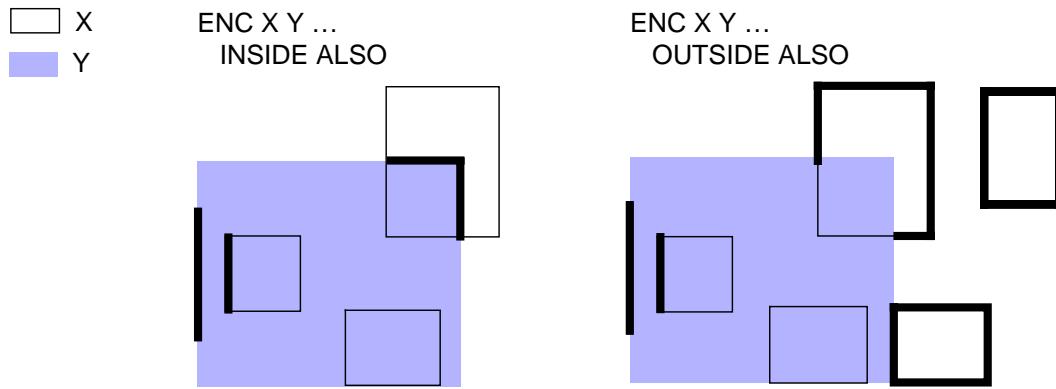
- *reversal*

Secondary keywords that instruct an ENCclosure operation to output *layer1* edges that are, or are not, enclosed by *layer2*, in addition to what is normally output.

**INSIDE ALSO** — Outputs edges from *layer2* that are enclosed by *layer1*. When you specify INSIDE ALSO, then edges from *layer2* that are inside *layer1*, but not coincident inside with *layer1* edges, are output by the operation (see Figure 4-90). The *layer1* parameter cannot be a derived edge layer.

**OUTSIDE ALSO** — Outputs edges from *layer1* that are not enclosed by *layer2*. When you specify OUTSIDE ALSO, then edges from *layer1* that are outside *layer2*, or coincident outside with *layer2* edges, are output by the operation (see Figure 4-90). If *layer2* is a derived edge layer, then the semantics are restricted to coincident outside edges from *layer1*.

**Figure 4-90. INSIDE ALSO and OUTSIDE ALSO**



The INSIDE ALSO and OUTSIDE ALSO keywords produce one-edge clusters if output is to an error layer; all other error-directed output from a dimensional check operation consists of two-, three-, or four-edge clusters (see “Edge Cluster Generation” in the *Calibre Verification User’s Manual* for details). This behavior is transparent to an edge-directed dimensional check operation. The edge is simply output according to the layer from which it originated. Also, these keywords create output in the absence of any measurement process by the operation. That is, output comes from an edge topological operation perspective rather than from a dimensional check operation perspective.

The INSIDE ALSO and OUTSIDE ALSO keywords are useful in eliminating time-consuming operations. Consider the following design rule:

Contacts must be totally enclosed by metal by at least 2 microns.

If you do not use the OUTSIDE ALSO keyword, this check requires the two following operations for complete accuracy:

```
// Spacing and touching.
rule_p {
ENCLOSURE contact metal < 2 ABUT == 0 SINGULAR
NOT INSIDE contact metal // contacts not totally enclosed
}
```

If you do use the OUTSIDE ALSO keyword, you can eliminate the Not Inside operation, as follows:

```
rule_q {
ENCLOSURE contact metal < 2 ABUT == 0 SINGULAR OUTSIDE ALSO
}
```

Notes concerning the INSIDE ALSO and OUTSIDE ALSO keywords:

- Because there is no measurement involved in generating output specifically from INSIDE ALSO and OUTSIDE ALSO keywords, the polygon containment criteria, metrics, and orientation keywords do not apply.
- Connectivity keywords do not influence output from the OUTSIDE ALSO keyword in an ENCclosure operation.
- If you specify NOT CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it is on the same electrical node. If you specify CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it is on a different electrical node.
- In the previous examples, the tool can produce spurious errors. In the ENCclosure check, four errors are produced for every isolated rectangular contact. Such errors are grouped by switching to a polygon-directed form (see “[region\\_output](#)” on page 528).
- It is not necessary to use the OVERLAP keyword with either INSIDE ALSO or OUTSIDE ALSO if you intend to find shared or non-shared areas. This can result in false errors.
- EXCLUDE FALSE

A secondary keyword used to eliminate rare false errors due to the absence of endpoint blocking edges at the correct hierarchical level when two edges are measured. This is often referred to as a false notch. This only applies to hierarchical runs. This keyword should only be used if false errors are actually detected and they cannot be tolerated. This option has a considerable runtime penalty.

See “[False Measurement Reduction](#)” in the *Calibre Verification User’s Manual* for more details.

- *region\_output*

An optional keyword that instructs dimensional check operations to generate a derived edge or polygon layer instead of a derived error layer. (Derived edge and polygon layers can be further manipulated by other operations, whereas derived error layers cannot.) For detailed information on this subject, see “Polygon-Directed Output” in the *Calibre Verification User’s Manual*.

The value of output takes the following form:

REGION [EXTENTS | CENTERLINE [*value*]]

REGION — Constructs edge projections between the endpoints of selected edges to create polygonal regions; the composite of the selected edges and edge projections is output as derived polygon data.

If you are deriving polygon layers, using the REGION keyword with the Euclidean (default) metric can generate large numbers of skew edges, which must be flattened in hierarchical applications. This is usually wasteful and requires much excess processing time. REGION EXTENTS is usually the best choice for deriving polygon layers. You can also use the OPPOSITE metric to eliminate skew edges, but be aware that you can miss legitimate design rule errors (such as corner-to-corner configurations) when using OPPOSITE.

Skew output edges resulting from the REGION keyword require special handling. The final placement of such edges is the result of merging and numeric rounding. This can result in the placement of skew edge vertices that differ up to 1 dbu from the layout geometry.

REGION EXTENTS — Constructs derived polygon data as for REGION, but the output is the rectangular extents of the polygons output by REGION, rather than the polygons themselves. The tool forms the extents prior to the merging of the regions.

---

**Note**



We recommend using REGION EXTENTS in most layer derivation applications. You should *not* use REGION EXTENTS when polygonal regions are already orthogonal to the database axes, such as those created with the OPPOSITE metric.

---

REGION CENTERLINE [*value*] — Constructs derived polygon data as for REGION. The output consists of the centerlines of the polygonal regions, rather than the regions themselves. These centerlines are formed prior to the merging of the regions. The centerlines are along the direction of the edges whose measurement forms the region; they have a default width of eight database units. The optional parameter *value* allows you to specify the centerline width. The *value* must be a floating-point number greater than or equal to two database units.

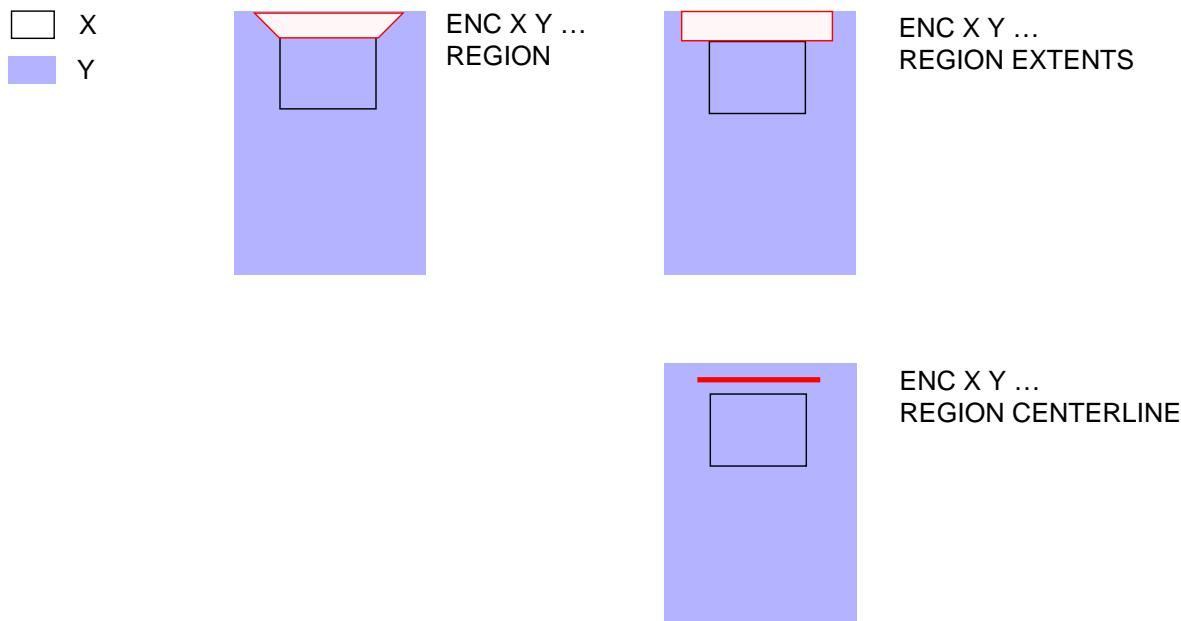
For two, parallel, horizontal edges, the y-coordinate of the centerline segment is always closer to the bottom edge if any snapping occurs. The formula for the y-coordinate of the centerline is:

$$y1 + ( ( y2 - y1 ) / 2 )$$

For vertical edges, a similar change keeps the x-coordinate of the line segment closer to the left edge.

Figure 4-91 shows examples of these keyword sets.

**Figure 4-91. REGION Examples**



## Examples

For additional examples, see “[Enclosure and Extension Checks](#)” on page 2089 and “[Contact Checks](#)” on page 2096.

## ERC Cell Name

Specification statement

### **ERC CELL NAME NO**

### **ERC CELL NAME YES [CELL SPACE] [XFORM] [ALL]**

Used only in Calibre nmLVS-H.

#### **Parameters**

- **NO**

Keyword that specifies not to append the cell instance name associated with each ERC result to its signature line in the ERC results database. This is the default behavior when you do not include this statement in the rule file.

- **YES [CELL SPACE] [XFORM] [ALL]**

Keyword, followed by an optional keyword set, that specifies to append the cell name associated with each ERC result to its signature line in the ERC results database. When you specify YES, you can also specify any of the following optional keywords:

**CELL SPACE** — Specifies to output the coordinates of the ERC result in the coordinate space of the cell in which it is instantiated, rather than top-level cell space. Top-level space is the default.

**XFORM** — Specifies to append the ERC result's transformation information to its signature line in the ERC results database.

**ALL** — Specifies to list the cell name and transformation information after every ERC result in the database.

#### **Description**

Specifies whether to append the cell name associated with each ERC result to the associated signature line in an ERC results database. The default is to specify results in top-cell coordinate space, with no cell name information appended to the ERC results.

This statement applies to ASCII ERC results databases only and can be specified at most once.

If you specify YES, then the cell name follows the signature for the individual ERC result in the results database as follows:

```
p <error-number> <vertex-count> <cell-name>
```

```
e <error-number> <edge-count> <cell-name>
```

Each cell has a unique transform associated with it which represents the to-world transform of exactly one of its placements throughout the entire hierarchy. This transform is used consistently for all ERC results from that cell.

If the CELL SPACE keyword is specified, then all results have their coordinates output in the coordinate space of the cell, not the top-level space. So that database readers can determine

which space the coordinates are actually output in, the CELL SPACE keyword causes a space followed by a ‘c’ character to be written after the cell name in the signature of the ERC result:

```
p 2 4 NAND034 c
```

If you specify the XFORM keyword, Calibre nmLVS-H appends the transformation information for each ERC result to the end of its signature line in the ERC results database. The transformation information is part of a  $3 \times 3$  matrix of six integers that represent the object's transformation from local cell space to top-cell space coordinates, if CELL SPACE has also been specified. Otherwise, it is the inverse transformation to cell-space coordinates from top-cell space coordinates (the default). Cell location coordinates are vectors  $(x, y, 1)$ , where  $x, y$  is a location of a point. Cell location vectors are multiplied by the  $3 \times 3$  transformation matrix to map the location coordinates into the appropriate space.

For example:

```
p 2 4 NAND034 c 0 1 -1 0 456547 377748
```

this gives ERC results where the numbers after the *c* character represent the transformation matrix information.

In this example, the row vectors of the matrix would be  $(\mathbf{0}_{00}, \mathbf{1}_{01}, 0_{02})$ ,  $(-\mathbf{1}_{10}, \mathbf{0}_{11}, 0_{12})$ ,  $(\mathbf{456547}_{20}, \mathbf{377748}_{21}, 1_{22})$ . The bold numbers correspond to the integers after the *c* character. Each cell has a unique set of transformation numbers that represents the transformation of exactly one of its placements in the hierarchy. Generally, this is the lowest, left-most placement of the cell. The transformation numbers are used consistently for all ERC results from that cell.

Matrix elements 00, 01, 10, and 11 control rotation and have values of 0, 1, or -1. Elements 20 and 21 are in database units. Elements 02 and 12 always have values of 0. Element 22 is always 1.

Floating-point transformation numbers are not supported. They are automatically removed from the input layout database by flattening or cloning. Calibre nmLVS-H does not list the cell's transformation data after multiple outputs of the same cell name.

The XFORM keyword is generally used for viewing results in local cell space in Calibre RVE. Calibre RVE uses the transformation data to allow ERC error highlighting in both the cell and top-level contexts. See “[Viewing Results in Cell Space](#)” in the *Calibre Verification User’s Manual* for more details.

By default, Calibre nmLVS-H minimizes the ASCII ERC results database size due to inclusion of the cell name and transformation data by omitting this information for any ERC result that is identical to that of the previous ERC result (with recurrence). The information is provided for the first ERC result in each ERC rule check. The ALL keyword overrides this default behavior.

See also [ERC Results Database](#).

## Examples

```
ERC CELL NAME YES CELL SPACE XFORM
// report in cell-space coordinates and show transformation
// matrix elements
```

## ERC Check Text

Specification statement

**ERC CHECK TEXT {{[COMMENTS | ALL]} [RFI]} | NONE**

### Parameters

- COMMENTS

Keyword that instructs the tool to output the rule check comments to the ERC results database. This is the default behavior, along with RFI.

- ALL

Keyword that instructs the tool to output the entire text of the rule check to the ERC results database.

- RFI

Keyword that instructs the tool to output the rule file title and pathname to the ERC results database. When the optional keyword COMMENTS is included, the tool also outputs the rule file title and pathname to the ERC results database. COMMENTS RFI is the default if you do not include this statement in the rule file. When the optional keyword ALL is included, the tool also outputs the rule file title and pathname to the ERC results database.

- NONE

Keyword that instructs the tool to not output any rule check text or rule file information to the ERC results database.

### Description

Specifies the amount and type of text to appear in ERC results databases.

You can specify this statement once in your rule file.

Text consists of at sign (@) comments, rule file information, and rule check text. This statement allows you to include some or all of the rule check text in the ERC results database.

See also [ERC Results Database](#).

### Examples

#### Example 1

```
// Default setting.  
ERC CHECK TEXT COMMENTS RFI
```

#### Example 2

```
// Place all ERC text in the ERC results DB.  
ERC CHECK TEXT ALL
```

#### Example 3

```
// No ERC text in the results DB.  
ERC CHECK TEXT NONE
```

## ERC Keep Empty

Specification statement

### ERC KEEP EMPTY {YES | NO}

#### Parameters

- **YES**

Keyword that instructs the tool to retain rule checks containing zero results in ERC execution and to write them to the ERC results database. This is the default if you do not include this statement in the rule file.

- **NO**

Keyword that instructs the tool not to output rule checks containing zero results to the ERC results database.

#### Description

Specifies whether empty rule checks (rule checks with zero results) are retained in ERC.

You can specify this statement once in your rule file.

This statement applies to ASCII ERC results databases. See also [ERC Results Database](#).

#### Examples

```
// Keep empty ERC results.  
ERC KEEP EMPTY YES
```

# ERC Maximum Results

Specification statement

## ERC MAXIMUM RESULTS {*number* | ALL}

### Parameters

- *number*

Non-negative integer that specifies the maximum result count for an individual rule check in ERC. When you do not include this statement in your rule file, the default value of *number* is 1000.

- **ALL**

Keyword that specifies for 32-bit platforms the maximum result count is  $2^{32}$ . For 64-bit platforms it is  $2^{64}$ .

### Description

Specifies the maximum number of results for an individual rule check in ERC.

You can specify this statement once in your rule file.

When the maximum results are generated for a rule check, the tool issues a warning and no further results are added to the ERC results database for that rule check.

You can specify ERC options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Running ERC Checks](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [ERC Results Database](#).

### Examples

#### Example 1

```
// Use default explicitly.  
ERC MAXIMUM RESULTS 1000
```

#### Example 2

```
// Send all results to the ERC DB for each check.  
ERC MAXIMUM RESULTS ALL
```

#### Example 3

```
// Send first 50 results to the ERC DB for each check.  
ERC MAXIMUM RESULTS 50
```

# ERC Maximum Vertex

Specification statement

**ERC MAXIMUM VERTEX {*number* | ALL}**

## Parameters

- ***number***

Non-negative integer that specifies the maximum vertex count of any ERC result polygon the tool writes to the ERC results database. The value must be an integer greater than or equal to 4. When you do not include this statement in your rule file, the default value of ***number*** is 4096.

- **ALL**

Keyword that specifies a  $2^{32}$  maximum vertex count for an output polygon.

## Description

Specifies the maximum vertex count of any ERC result polygon the tool writes to the ERC results database. You can specify this statement once in your rule file.

The tool segments ERC result polygons whose vertex count exceeds the specified value and writes the segmented polygons to the ERC results database. The segmented polygons have a vertex count less than or equal to the supplied value.

The tool reports a warning if polygon segmentation occurs, once per rule check, and the warning indicates the number of polygons segmented.

You can specify ERC options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Running ERC Checks](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [ERC Results Database](#).

## Examples

### Example 1

```
// Use default.
ERC MAXIMUM VERTEX 4096
```

### Example 2

```
// Segment results into quadrilaterals.
ERC MAXIMUM VERTEX 4
```

## ERC Path Also

Specification statement

### ERC PATH ALSO [BIPOLAR] [CAPACITOR] [DIODE]

#### Parameters

At least one of the following must be specified in this statement:

- **BIPOLAR**

Keyword that specifies bipolar transistors (element name Q) are included in the definition of “path” for [Pathchk](#) and [ERC Pathchk](#) statements.

- **CAPACITOR**

Keyword that specifies capacitors (element name C) are included in the definition of “path” for Pathchk and ERC Pathchk statements.

- **DIODE**

Keyword that specifies diodes (element name D) are included in the definition of “path” for Pathchk and ERC Pathchk statements.

#### Description

Enables the specified devices to be considered in the definition of “path” for ERC path checking. When this statement is specified in a rule file, a path goes through POS and NEG pins of diodes and capacitors, and through C and E pins of bipolar devices. Resistors and MOS devices are always included in the definition of path.

The definition of path for the purpose of determining unused devices in Pathchk and ERC Pathchk is similar to that used in circuit comparison and is not affected by ERC Path Also.

User-defined bipolar, capacitor, and diode devices mapped to built-in devices using the [LVS Device Type](#) statement also are included in the definition of path by this statement.

This statement may appear any number of times and keywords are accumulated from all individual statements. Keywords may be repeated and duplications are ignored.

#### Examples

```
ERC PATH ALSO CAPACITOR DIODE /* add capacitors and diodes to definition
of path */
rule_1.1 {PATHCHK !POWER && !GROUND}

ERC SELECT CHECK rule_1.1

LVS EXECUTE ERC YES
```

# ERC Pathchk

Specification statement

## ERC PATHCHK

```
{labeled_flag | power_ground_flag [operand power_ground_flag]}
[POLYGONS NETS | POLYGONS | NETS]
[BY LAYER] [BY CELL | FLAT]
[PORTS ALSO] [NOFLOAT] [EXCLUDE UNUSED]
[EXCLUDE SUPPLY]
```

Used during circuit extraction in Calibre nmLVS/nmLVS-H and ICtrace Mask mode.

### Note



This statement is provided primarily for backward compatibility. The [Pathchk](#) layer statement produces geometric output and is generally more useful for conducting ERC checks.

## Summary

This specification statement reports nets in the layout that do or do not have a path to power, ground, or labeled nets, or that satisfy a combination of these conditions. Power, ground, and labeled nets are top-level nets. This is performed as a part of connectivity extraction in LVS. This statement is *independent* of any of the other ERC operations and statements; hence, the name of this statement can be somewhat misleading. It has no interaction with [ERC Select Check](#) or [LVS Execute ERC](#), and it does not produce data that can be used in any layer operation.

## Parameters

- *labeled\_flag*

A required string that specifies how the tool reports a net that does or does not have a path to a labeled net. This option uses top-level text objects that are neither power nor ground net labels. Possible choices are:

**LABLED** — Reports nets with a path to a labeled (texted) net.

**!LABLED** — Reports nets with no path to any labeled (texted) net.

where the exclamation mark (!) is synonymous with Boolean NOT.

- *power\_ground\_flag*

A required string that specifies the type of net the tool does or does not report. Power and ground nets are top-level nets specified in the [LVS Power Name](#) and [LVS Ground Name](#) statement. Possible choices are:

**POWER** — Reports nets with a path to power.

**!POWER** — Reports nets with no path to power.

**GROUND** — Reports nets with a path to ground.

**! GROUND** — Reports nets with no path to ground.

Where the exclamation mark (!) is synonymous with Boolean NOT.

The *labeled\_flag* and the *power\_ground\_flag* keywords, when not connected by an *operand*, are called *primitive* conditions.

- *operand*

An optional string of logical symbols that combine two primitive conditions. Possible choices are:

&& — Selects nets that satisfy both primitive conditions: Boolean AND.

|| — Select nets that satisfy the first or second primitive condition, or both: Boolean OR.

- POLYGONS\_NETS | POLYGONS | NETS

Specifies the type of results file to output. Possible choices are:

POLYGONS — Generates an ASCII nmDRC file that contains the polygon output of all nets that satisfy the specified conditions.

NETS — Generates a text file that contains a list of all nets that satisfy the specified conditions.

Both are specified by default.

- BY LAYER [BY CELL | FLAT]

Optional keyword set that specifies the type of output to include in the generated ASCII results file. These pertain to the POLYGONS option only. Possible choices are:

BY LAYER — This option generates a separate check for each layer. If both BY CELL and BY LAYER are specified, a check is generated for each layer in each cell. Empty checks are not generated.

BY CELL — Causes each cell that has cell-specific results to be checked separately. The check contains nets reported in that cell. The results are reported in top-level coordinate space. BY CELL is ignored in flat execution.

FLAT — Causes cell-specific results to be replicated in every placement. FLAT has no effect in flat execution.

- PORTS ALSO

An optional keyword that specifies to include nets that are connected to port objects in the top cell in results output. By default, nets that are connected to port objects in the top cell are not included in the output for any request. The PORTS ALSO option removes the special treatment of port nets. This option affects both POLYGONS and NETS output. Note that port objects are specified with the [Port Layer Text](#) in Calibre, and the \$make\_port() command in Pyxis Layout.

- NOFLOAT

An optional keyword that instructs the statement to ignore floating nets. Floating nets are nets that are not connected to any devices.

- EXCLUDE UNUSED

An optional keyword that instructs the statement to exclude *unused* nets from the ERC Pathchk output. Unused nets are connected only to unused devices. Unused devices are specified with [LVS Filter Unused Option](#) or equivalent LVS Filter Unused statements. All options for LVS Filter Unused Option are supported except the INV, P, and Q options.

- EXCLUDE SUPPLY

An optional keyword that instructs the tool to exclude power and ground nets from the results of an ERC Pathchk request containing at least one of the keywords **POWER** or **GROUND** (not preceded by the ! operator). Supply nets are included if EXCLUDE SUPPLY is not specified.

## Description

Reports nets in the layout that do, or do not, have a path to top-level power nets, ground nets, labeled nets, or nets that satisfy a combination of these conditions.

By default, a *path* is any route that leads through source/drain pins of built-in MOS devices or pos/neg pins of built-in resistor devices. To qualify, a MOS device (types M, ME, MD, MN, MP, or any of the LDD-type devices) must have at least the following pins: G (or gate), S (or source), and D (or drain). Paths through LDD-type devices are formed in both directions.

Capacitor, diode, or bipolar devices can also be added to the path, as specified in [ERC Path Also](#) statements.

If you want to enable user-defined devices to participate in ERC Pathchk (user-defined devices do not participate by default), then you must map these devices and their pins to built-in devices and pins. This is done using the [LVS Device Type](#) specification statement. See “User-Defined Devices in Path Check Operations” in the [Calibre Verification User’s Manual](#).

Power and ground nets are specified with [LVS Power Name](#) and [LVS Ground Name](#) specification statements, respectively. This statement utilizes wildcard characters that appear in LVS Power Name and LVS Ground Name specification statements. Power and ground nets break paths.

Labeled nets are those with text, but they are not power or ground nets. Labeled nets typically belong to top-level I/O ports.

The hierarchical level from which text objects are read is controlled by the [Text Depth](#) statement.

Power and ground nets are included in the result for the keywords **POWER** and **GROUND**. For example, supply nets are included if you specify:

### **ERC PATHCHK POWER**

You can exclude supply nets by specifying EXCLUDE SUPPLY. When specifying EXCLUDE SUPPLY, at least one of the **POWER** or **GROUND** keywords must also be specified in the statement. (The EXCLUDE SUPPLY keyword does not apply to the ! **POWER** or ! **GROUND** keywords.) For example:

### **ERC PATHCHK POWER && !GROUND EXCLUDE SUPPLY**

is allowed. However,

**ERC PATHCHK !POWER && !GROUND EXCLUDE SUPPLY // bad syntax**

causes a compiler error.

Power and ground nets are not included in the result for **!POWER** and **!GROUND** keywords. For example, the supply net itself is not selected by this:

**ERC PATHCHK ! GROUND**

even if there is no path from power to ground.

If a net in the design matches more than one of LVS Power Name or LVS Ground Name name patterns, a warning is issued and conflicts are resolved with power names being given priority over ground names.

The **LABLED** option does not consider supply nets as labeled.

ERC Pathchk statements respect the [LVS Compare Case](#) statement. If either the LVS Compare Case YES or LVS Compare Case NAMES statement is specified, then power names and ground names are traced in a case-sensitive manner; otherwise they are case insensitive. These rules on case sensitivity apply to LVS Power Name and LVS Ground Name nets.

ERC Pathchk observes the [Layout Preserve Case](#) specification statement for case-sensitivity of net names. See “[Case Sensitivity of Power and Ground Names](#)” in the *Calibre Verification User’s Manual* for more details.

This statement applies to Calibre nmLVS/nmLVS-H and ICtrace Mask mode. It is executed during circuit extraction and only when the layout is geometric. The following Calibre commands execute this statement:

```
calibre -lvs rules           // executed flat
calibre -lvs -tl ... rules   // executed flat
calibre -lvs -hier ... rules // executed hierarchically with
                             // geometric Layout System
calibre -spice ... rules     // executed hierarchically
```

The following Pyxis Layout commands execute this statement:

```
$lvs_mask()      // executed flat
$write_mask_cnet() // executed flat
```

Note the following important considerations:

- ERC Pathchk operates only on nets connected to devices. It does not report floating nets.
- The **POWER**, **GROUND**, and **LABLED** options (without the **!** operator) are not recommended because they can generate very large files, consume large amounts of memory, and lead to long runtimes. In particular, in hierarchical ERC, such requests essentially flatten the design hierarchy.

- In the default mode, with neither FLAT nor BY CELL specified, the polygon output may contain polygons belonging to a cell in two ways:
  - If a net in the cell violates ERC Pathchk requirements in some placements of this cell, the polygons are written out for each placement where the violation occurs in top-level coordinates.
  - If, for a given net, a violation appears in every placement of a cell, the polygons are written out once, in the lowest, leftmost placement.

This can lead to confusion because, in the same cell, some polygons may be displayed in many placements and others in only one placement. If the results are difficult to interpret, the BY CELL option can be useful.

The FLAT option can also be used to force Calibre to output cell-specific results in all placements. This increases both memory consumption and the size of the polygon file, but can eliminate potential confusion caused by hierarchical output. However, output using the NETS option is unambiguous in all cases.

ERC Pathchk statements are not executed if the specified net is not present. For example, if there is no power net in the layout, then ERC PATHCHK ! POWER generates an error message and generates no output.

## Results Output Options

The output files are readable by RVE similar to a DRC Results Database.

**POLYGONS** — The POLYGONS option specifies a file named *layout\_primary*.pathchk.erc. It is placed in the [Mask SVDB Directory](#) if that statement is specified, otherwise in the current working directory. If *layout\_primary* is not specified, the word lvs is used instead. The file consists of lines in the format:

```
ERC PATHCHK condition [ in cell cell_name ] [ layer_name ]
```

where *condition* is the specified condition, such as ! POWER or ! POWER && GROUND. By default, results from each ERC Pathchk statement are written to one check in the result file. If the BY CELL option is specified, then the results are classified by cell and the *cell\_name* field indicates the cell name. If the BY LAYER option is specified, then the results are classified by layer and the *layer\_name* field indicates the layer. Empty checks are not written. If all checks are empty, no file is created.

All polygon results are shown in the top-level coordinate space, and they are written in nmDRC ASCII format. By default, hierarchical Calibre shows cell-specific results only once in a representative placement of the cell. The lowest, leftmost placement of each cell serves as its representative. This is similar to Calibre nmDRC-H. Results are cell-specific if they can be generated locally in the cell and occur in all placements of the cell in the design.

Note that polygon results generated by ERC Pathchk are not fully merged. This means that polygons can abut or overlap.

**NETS** — The NETS option specifies a file named *layout\_primary*.pathchk.rep. It is placed in the SVDB directory by default, otherwise in the current working directory. If *layout\_primary* is not specified, the word lvs is used instead.

In hierarchical execution, the report file format is this:

```
ERC PATHCHK REPORT for layout_primary
ERC PATHCHK condition:
  cell cell_name (placement): net1, net2, ...
  cell cell_name (placement): net1, net2, ...
```

where *condition* is the specified condition, such as GROUND AND ! POWER, *cell\_name* is a cell name where results were found, *placement* is a representative placement of that cell, and *neti* is a net name or number. Cells are listed in bottom-up order. Net names and numbers are in the context of the indicated cells. Cells for which no nets are found by ERC Pathchk are not written to the report file.

In flat execution, the report file format is:

```
ERC PATHCHK REPORT for layout_primary
ERC PATHCHK condition:
  net1, net2, ...
```

Nothing is reported for requests in which no nets are found in any cell. If all checks are empty, no file is created.

## Requirements and Restrictions

Hierarchical Calibre operations impose a size limit on intermediate data structures generated by ERC Pathchk. The limit is approximately 100 MB on 32-bit platforms and 400 MB on 64-bit platforms. This limit is applied independently to each individual ERC Pathchk statement. Calibre generates incomplete results (some polygons may be omitted from the output) and issues the following warning if Calibre reaches this limit:

```
WARNING: ERC operation exhausted allowed memory allocation.
```

Calibre issues the warning in both the transcript and the circuit extraction report.

This limit applies only to polygon output, for example during the construction of a result layer and when performing the POLYGONS option.

Calibre always completes the net reporting part of this functionality, including the functionality resulting from the NETS option.

You should not reach this limit in normal operations. You see it only in circumstances when a very large net is selected by the ERC Pathchk statement.

See also [ERC Summary Report](#). The search features [Find Path](#) and [Isolate Path on Layout Net](#) in Calibre RVE for LVS may also be of interest.

## Examples

### Example 1

The following statement outputs a text file listing all the nets with a path to a labeled net:

```
ERC PATHCHK LABELED NETS
```

### Example 2

When not using the [ ! ] LABELED option, the following combinations are possible:

**! POWER && GROUND**

Nets with no path to power but with a path to ground

**! GROUND && POWER**

Nets with no path to ground but with a path to power

**! GROUND && ! POWER**

Nets with no path to ground and no path to power

**! GROUND || ! POWER**

Nets with no path to ground or no path to power (or both)

**POWER && GROUND**

Nets with a path to both power and ground

**POWER || GROUND**

Nets with path to power or a path to ground

# ERC Results Database

Specification statement

**ERC RESULTS DATABASE** *filename* [ASCII] [PSEUDO]

## Parameters

- *filename*

A required pathname where the tool writes the ERC results database.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

- ASCII

An optional keyword that indicates the ERC results database is in ASCII format. This is the default behavior if you do not include ASCII in this rule file statement.

- PSEUDO

An optional keyword that specifies to generate a results database that includes pseudocells generated during hierarchical injection. Only applies to Calibre nmLVS-H.

## Description

Specifies the pathname and type of ERC results database generated during LVS connectivity extraction. You can specify this statement only once in your rule file and is required for ERC execution in Calibre nmLVS and mask-mode ICtrace. See the “Electrical Rule Checks” chapter in the [Calibre Verification User’s Manual](#) for more information on ERC. The format for the ERC results database is discussed in the section “[ASCII nmDRC Results Database Format](#)” in the same manual.

When you specify PSEUDO in this rule file statement, Calibre includes the additional cells that Calibre creates to support its hierarchical algorithms. The cells are named ICV\_*n*, where *n* is a unique integer.

If you do not specify PSEUDO, Calibre does not include the pseudocells in the ERC results database and flattens the data residing within the pseudocells up to the next level of user hierarchy. For ASCII databases, transformation to the top-level proceeds normally.

Limitations on output file size are operating-system dependent. In 32-bit mode, the maximum practical file size is 2 GB. If you anticipate output file sizes near this value, you should use 64-bit mode.

You can specify ERC options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Running ERC Checks](#)” in the [Calibre Interactive and Calibre RVE User’s Manual](#).

See also [LVS Execute ERC](#), [ERC Maximum Results](#), [ERC Maximum Vertex](#), [ERC Select Check](#), and [ERC Unselect Check](#).

## Examples

The following statement outputs an ASCII format of the results database to the specified location. You can use Calibre RVE to analyze the results or bring the results into Pyxis Layout for viewing.

```
ERC RESULTS DATABASE "~/design1/drc_database" ASCII
```

## ERC Select Check

Specification statement

**ERC SELECT CHECK [ABORT LVS] *rule\_check* [*rule\_check* ...]**

### Parameters

- **ABORT LVS**

An optional keyword that terminates nmLVS before the circuit comparison step if any *rule\_check* in the statement generate non-empty results. If used, this keyword must appear before any *rule\_check* names. This keyword is used only with Calibre nmLVS/nmLVS-H.

- ***rule\_check***

A required name of a rule check statement or rule check group from the rule file. You can specify *rule\_check* any number of times in one statement.

### Description

Selects rule check statements and rule check groups to be executed by ERC during LVS. (Because the [Inductance MICheck](#) rule requires information from Calibre xRC, rule check statements containing the inductance rule are ignored.) While any layer operation can appear in an ERC rule check, the primary operation of interest for ERC rule checks is [Pathchk](#). A Calibre nmDRC product license is also used during ERC.

---

**Note**



This statement is required for ERC results to be generated in LVS. For rule checks that may also be run in nmDRC, the nmDRC tool automatically unselects all rule checks in the rule file that are specified in ERC Select Check statements, unless such rule checks are also explicitly specified in a [DRC Select Check](#) statement.

---

You can specify this statement any number of times in your rule file.

The following algorithm describes how LVS applications (Calibre nmLVS and mask-mode ICtrace) select rule checks:

1. Unselect all rule checks.
2. Select rule checks specified in ERC Select Check statements.
3. Unselect rule checks specified in [ERC Unselect Check](#) statements.

In contrast the following algorithms describe how Calibre nmDRC/nmDRC-H selects rule checks:

1. If DRC Select Check statements are in the rule file:
  - a. Unselect all rule checks.
  - b. Select rule checks specified in DRC Select Check statements.
  - c. Unselect rule checks specified in [DRC Unselect Check](#) statements.

2. If DRC Select Check statements are *not* in the rule file:
  - a. Select all rule checks.
  - b. Unselect rule checks specified in ERC Select Check statements.
  - c. Unselect rule checks specified in DRC Unselect Check statements.

If you specify the ABORT LVS option, and if any **rule\_check** in the statement returns non-empty results, then nmLVS terminates after performing connectivity extraction, device recognition, and ERC operations, but before the circuit comparison step. All errors and warnings generated by connectivity extraction are reported normally. This behavior occurs if **LVS Abort On ERC Error YES** is specified either explicitly or by default.

The same **rule\_check** may be selected by more than one ERC Select Check statement. In this case, if at least one of these statements has ABORT LVS specified, then the **rule\_check** is used to terminate nmLVS.

If a **rule\_check** generates non-empty results, and ABORT LVS is specified, Calibre prints the following message in the transcript immediately after running the check:

```
WARNING: Results generated by an ERC check with ABORT LVS (check <check name>) - aborting.
```

This message is printed once for every check that activates the abort on ERC error. The following message is also printed once at the end of the transcript:

```
ERROR: Results generated by one or more ERC checks with ABORT LVS - aborting.
```

Any ERC operations that generate their own results files instead of, or in addition to, result layers do not trigger ABORT LVS termination for ERC errors. For example, consider the following ERC check that uses the PRINT POLYGONS option:

```
ERC SELECT CHECK ABORT LVS CHECK1
CHECK1 {
  X = PATHCHK !POWER PRINT POLYGONS "nopower.rdb"
  X AND EMPTY_LAYER
}
```

This check does not abort nmLVS even if the Pathchk operation generates results in the nopower.rdb database because the check itself is guaranteed to be empty (as long as EMPTY\_LAYER is indeed empty).

ERC Select Check statements that use the ABORT LVS option can have the nmLVS termination feature globally disabled in the rule file by specifying **LVS Abort On ERC Error NO**. The ERC checks specified in such ERC Select Check Statements are executed during the run, but nmLVS does not terminate if there are non-empty results. No special messages appear in the transcript if LVS Abort On ERC Error NO is specified.

You can specify ERC options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Running ERC Checks](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [ERC Results Database](#), [Group](#), and [LVS Execute ERC](#).

### Examples

#### Example 1

```
LVS EXECUTE ERC YES  
ERC SELECT CHECK PWR1 PWR2 PWR3
```

#### Example 2

```
// Abort LVS if PWR1, PWR2 or PWR3 returns non-empty results  
LVS EXECUTE ERC YES  
ERC SELECT CHECK ABORT LVS PWR1 PWR2 PWR3
```

# ERC Summary Report

Specification statement

**ERC SUMMARY REPORT** *filename* [REPLACE | APPEND] [HIER]

## Parameters

- ***filename***  
Required pathname of the summary report file.
- **REPLACE | APPEND**  
Optional keyword that controls the method in which the summary report file is opened.  
  - REPLACE** — Overwrites the previous contents of the summary file, if any, each time you run ERC. This is the default behavior if you do not specify a keyword from this set.
  - APPEND** — Appends to an existing summary report file. This parameter is useful in creating a history log of runs in a single file.
- **HIER**  
Optional keyword that instructs Calibre nmLVS-H to create a section in the summary report that lists non-empty rule check statistics by layout database cell. This keyword is used only in Calibre nmLVS-H.

## Description

This statement causes generation of an ERC Summary Report in Calibre nmLVS/nmLVS-H or ICtrace. The report is generated if ERC is executed. More precisely, the report is generated if an **ERC Pathchk** operation is present, or if an **ERC Select Check** specification statement is present and **LVS Execute ERC YES** is specified. The report is generated after the circuit extraction step.

This report information includes (but is not limited to):

- The date and time of the execution.
- The rule file pathname and title, if specified.
- The layout path, as appropriate, and top-level layout cell name.
- The current working directory and user name.
- The number of original shapes processed per original layer (excluding Pins, Ports, and Topex shapes for ICtrace).
- A summary of the number of results generated per ERC rule check, as well as an indication of which rule checks from the rule file were not executed.
- A summary of the number of results generated per **ERC Pathchk** statement.
- All warnings generated.
- The total run time and number of original shapes processed.

- Number of ERC rule checks executed, number of results generated as a result of the rule checks, number of results caused by PRINT [POLYGON | NET] options to [Pathchk](#) specification statements.
- Number of ERC Pathchk operations executed, number of NET and POLYGON results generated as a result of the ERC Pathchk operations.

You can specify ERC options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Running ERC Checks](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

### Examples

```
ERC SUMMARY REPORT erc.log APPEND HIER
//create a running log file for ERC executions
```

# ERC Unselect Check

Specification statement

**ERC UNSELECT CHECK *rule\_check* [*rule\_check* ...]**

## Parameters

- ***rule\_check***

A name of a rule check or rule check group from the rule file. You can specify ***rule\_check*** any number of times in one statement.

## Description

Unselects ERC rule check statements and rule check groups specified in this statement during LVS.

You can specify this statement any number of times in your rule file.

The following algorithm describes how LVS applications (Calibre nmLVS and mask-mode ICtrace) select rule checks:

1. Unselect all rule checks.
2. Select rule checks specified in [ERC Select Check](#) statements.
3. Unselect rule checks specified in ERC Unselect Check statements.

You can specify ERC options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Running ERC Checks](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [Group](#) and [LVS Execute ERC](#).

## Examples

```
LVS EXECUTE ERC YES
GROUP ALL_CHECKS PWR1 PWR2 PWR3 PWR4
ERC SELECT CHECK ALL_CHECKS

ERC UNSELECT CHECK PWR1 PWR2
// Exclude PWR1 and PWR2 ERC checks in the run
```

## Exclude Acute

Specification statement

**EXCLUDE ACUTE** *layer* [*layer* ...]

Used only in Calibre.

### Parameters

- *layer*

A required name (not number) of an original layer. You can specify *layer* any number of times in one statement.

### Description

Identifies layers to be excluded from [Flag Acute](#) and [Drawn Acute](#) checking. If Flag Acute YES is specified or Drawn Acute is executed, any object on a layer in an Exclude Acute specification statement is not checked.

Since polygon flagging is per original layer (not simple layer), if *layer* is a layer set, it is not equivalent to its constituent simple layers listed independently.

See also [Exclude Angled](#) and [Exclude Skew](#).

### Examples

```
// Do not check metall1 and metall2 for acute angles.  
EXCLUDE ACUTE metall1 metall2
```

## Exclude Angled

Specification statement

**EXCLUDE ANGLED** *layer* [*layer* ...]

Used only in Calibre.

### Parameters

- *layer*

A required name (not number) of an original layer. You can specify *layer* any number of times in one statement.

### Description

Identifies layers to be excluded from [Flag Angled](#) and [Drawn Angled](#) checking. If Flag Angled YES is specified or Drawn Angled is executed, any object on a layer in an Exclude Angled specification statement is not checked.

Since polygon flagging is per original layer (not simple layer), if *layer* is a layer set, it is not equivalent to its constituent simple layers listed independently.

See also [Exclude Acute](#) and [Exclude Skew](#).

### Examples

```
// Do not check metall1 and metall2 for angled geometry.  
EXCLUDE ANGLED metall1 metall2
```

## Exclude Cell

Specification statement

**EXCLUDE CELL** *cell\_name* [*cell\_name* ...]

### Parameters

- *cell\_name*

A required name of a cell. You can specify *cell\_name* any number of times in one statement. The *cell\_name* can be a string variable (see [Variable](#)). The cell name parameters can contain one or more asterisk (\*) wildcard characters, where the (\*) matches zero or more characters. When using the (\*) character, enclose the cell name in quotes. Otherwise, a compilation error occurs because the asterisk is a reserved symbol.

### Description

Specifies one or more cells to exclude from a geometric layout database—that is, layout cells that are not to be processed by the verification application. Excluding a cell means that no objects from any placement of the cell are processed. This includes all hierarchy within the placement. By default, no cells are excluded.

This specification statement functions in the Mask verification set and may be specified any number of times.

In Calibre applications, cell exclusion is not supported for binary, ASCII, or SPICE input database formats. Excluded cells are listed in the run transcript.

In ICVerify applications, cell exclusion is supported by the exclude switch to the \$check\_drc() function. For the initial session CHEck DRc command, there are no excluded cells by default unless the loaded rule file has a non-empty set of Exclude Cell specification statements. In that event, the defaults are taken from the rule file. For subsequent CHEck DRc commands, the -EXCLUDE <vector> option defaults to its last value unless a rule file has been loaded since the last CHEck DRc command and that rule file has a non-empty set of Exclude Cell specification statements. In that case, the defaults are again be taken from the rule file.

### Examples

```
// Do not process these cells for the run.  
EXCLUDE CELL mux2 mem_core
```

## Exclude Offgrid

Specification statement

**EXCLUDE OFFGRID** *layer* [*layer* ...]

Used only in Calibre

### Parameters

- *layer*

A required name (not number) of an original layer. You can specify *layer* any number of times in one statement.

### Description

Identifies layers to be excluded from [Flag Offgrid](#) and [Drawn Offgrid](#) checking. If Flag Offgrid YES is specified or Drawn Offgrid is executed, any object on a layer in an Exclude Offgrid specification statement is not checked.

Since polygon flagging is per original layer (not simple layer), if *layer* is a layer set, it is not equivalent to its constituent simple layers listed independently.

### Examples

```
// Do not check metal1 or metal2 for offgrid vertices.  
EXCLUDE OFFGRID metal1 metal2
```

## Exclude Skew

Specification statement

**EXCLUDE SKEW** *layer* [*layer* ...]

Used only in Calibre.

### Parameters

- *layer*

A required name (not number) of an original layer. You can specify *layer* any number of times in one statement.

### Description

Identifies layers to be excluded from [Flag Skew](#) and [Drawn Skew](#) checking. If Flag Skew YES is specified or Drawn Skew is executed, any object on a layer in an Exclude Skew specification statement is not checked.

Since polygon flagging is per original layer (not simple layer), if *layer* is a layer set, it is not equivalent to its constituent simple layers listed independently.

See also [Exclude Acute](#) and [Exclude Angled](#).

### Examples

```
// Do not check metall1 and metall2 for skew edges.  
EXCLUDE SKEW metall1 metall2
```

# Expand Cell

Specification statement

**EXPAND CELL** *cell\_name* [*cell\_name* ...]

Used only in hierarchical Calibre.

## Parameters

- *cell\_name*

A required name of a cell. You can specify *cell\_name* any number of times in one statement. The *cell\_name* can be a string variable (see [Variable](#)). The cell name parameters can contain one or more asterisk (\*) wildcard characters, where the (\*) matches zero or more characters. When using the \* character, enclose the cell name in quotes. Otherwise, a compilation error occurs because the asterisk is a reserved symbol.

## Description

Specifies the cells whose instances are to be expanded one level into the space of the cells in which the expanded instances are placed. Any underlying hierarchy of an expanded cell remains intact.

This statement is normally used to improve performance in gate arrays by expanding base cell containers down to the level of the base.

Text is not expanded by this statement. See [Expand Text](#) or [Expand Cell Text](#) (Calibre nmLVS-H and Calibre xRC only).

This statement can be specified any number of times.

Expand Cell is not intended to compensate for differences between source and layout hierarchy in nmLVS-H. Proper management of your hcell list avoids any need to use Expand Cell in such cases. Unnecessary use of Expand Cell can cause serious performance degradation in nmLVS-H.

## Examples

```
// Expand these cells one level.  
EXPAND CELL cella cellb
```

## Expand Cell Text

Specification statement

**EXPAND CELL TEXT** *origin\_cell* [*target\_cell* | PRIMARY]

Used only in hierarchical Calibre applications.

### Parameters

- *origin\_cell*

A required name of a cell whose text is to be expanded. The asterisk (\*) wildcard is allowed and matches zero or more characters. When using this wildcard, enclose the cell name in quotes. This parameter may also be a single-valued string [Variable](#).

- *target\_cell*

An optional name of a cell to which the text of the origin cell is expanded. The asterisk (\*) wildcard is allowed and matches zero or more characters. When using this wildcard, enclose the cell name in quotes. This parameter may also be a single-valued string [Variable](#).

- PRIMARY

An optional keyword that is equivalent to specifying the name of the primary cell.

### Description

Specifies that text from placements of the specified origin cell is to be used higher up in the layout hierarchy.

In the absence of a *target\_cell* or PRIMARY argument, the tool uses text from *origin\_cell* placements in any cell that directly contains such placements. The tool uses text from *origin\_cell* placements in the *target\_cell* if you specify *target\_cell* or PRIMARY. Participating *origin\_cell* placements can reside in the *target\_cell* or anywhere in the target cell's sub-hierarchy. An *origin\_cell* can contribute text to different *target\_cells* and a *target\_cell* can receive text from different *origin\_cells*.

This statement does not remove text from the *origin\_cell* but does add text to the specified *target\_cell* (whether specified implicitly or explicitly). The tool transforms text objects to the target cell's coordinate space and replicates them when multiple placements of *origin\_cell* are present.

The Expand Cell Text statement operates on connectivity extraction text and port text, but does not affect [With Text](#) operations. You must use this statement, rather than the [Expand Cell](#) statement, to preserve connectivity extraction text or port text. Used in nmDRC-H only if [DRC Cell Text YES](#) is specified.

This statement runs prior to Expand Cell statements and other implicit cell expansion statements. You can specify it any number of times.

This statement runs whether or not the specified original cell itself is expanded.

## Examples

### Example 1

In the following example, cell\_T1 contributes text to any cells that directly contain cell\_T1 placements. In addition, cell\_A and cell\_B receive text from cell\_T1 placements anywhere in their subhierarchy. cell\_B also receives text from cell\_T2 placements anywhere in its subhierarchy.

```
EXPAND CELL TEXT cell_T1
EXPAND CELL TEXT cell_T1 cell_A
EXPAND CELL TEXT cell_T1 cell_B
EXPAND CELL TEXT cell_T2 cell_B
```

### Example 2

In the following example, cells with names starting with TXT contribute text to any cells that directly contain them:

```
EXPAND CELL TEXT "TXT*"
```

### Example 3

In the following example, cells with names starting with TXT contribute text to cells with names starting with BLOCK. Participating TXT\* placements can reside in BLOCK\* cells or in their subhierarchy.

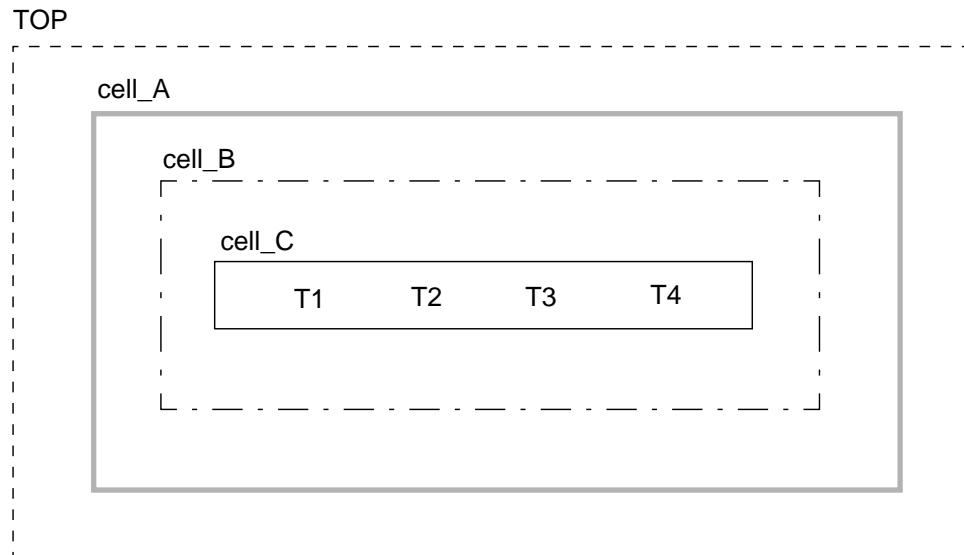
```
EXPAND CELL TEXT "TXT*" "BLOCK*"
```

### Example 4

In Figure 4-92, text objects T1, T2, T3 and T4 are in cell\_C and placed in cell\_A. They are not used in cell\_B or TOP.

```
EXPAND CELL TEXT cell_C cell_A
```

**Figure 4-92. Expand Cell Text cell\_C cell\_A**



## Expand Edge

Layer operation

### EXPAND EDGE *layer expansion\_set*

[EXTEND BY [FACTOR] *number*] [CORNER FILL]

#### Summary

Expands edges into rectangles in the specified directions and amounts.

#### Parameters

- *layer*

An original layer or layer set, or a derived polygon or edge layer.

- *expansion\_set*

A required keyword set that converts all edges of *layer* into rectangles by expanding the edges in the direction specified. The keyword sets are these:

**INside BY *value*** — Expands an edge towards the inside of the polygon it originated from by the amount specified in the *value* parameter. The *value* is in user units. Cannot be used with INside BY FACTOR.

**INside BY FACTOR *factor*** — Expands an edge towards the inside of the polygon it originated from by the product of the *factor* parameter and the edge's length. The *factor* is a non-negative real number. Cannot be used with INside BY.

**OUTside BY *value*** — Expands an edge towards the outside of the polygon it originated from by the amount specified in the *value* parameter. The *value* is in user units. Cannot be used with OUTside BY FACTOR.

**OUTside BY FACTOR *factor*** — Expands an edge towards the outside of the polygon it originated from by the product of the *factor* parameter and the edge's length. The *factor* is a non-negative real number. Cannot be used with OUTside BY.

**BY *value*** — Shorthand notation for specifying both keywords INside BY and OUTside BY. You cannot use this keyword with any other keyword in this set.

**BY FACTOR *factor*** — Shorthand notation for specifying both keywords INside BY FACTOR and OUTside BY FACTOR. You cannot use this parameter with any other keyword in this set.

- EXTEND BY [FACTOR] *number*

An optional keyword set that directs an Expand Edge operation to extend or retract the length of all *layer* edges prior to expanding them into rectangles. The parameter *number*, which is a positive or negative real number, must follow the optional keyword you specify for this set. Possible choices are as follows:

**EXTEND BY *number*** — A positive *number* parameter specifies the amount to extend both ends of an edge. A negative *number* parameter specifies the amount to retract both

ends of an edge. An edge is not generated if *number* is greater than or equal to half the original edge length.

EXTEND BY FACTOR *number* — The product of a positive *number* parameter and an edge's length specifies the amount to extend both ends of the edge. The product of a negative *number* parameter and an edge's length specifies the amount to retract both ends of an edge. An edge is not generated if *number* is greater than or equal to half the original edge length.

- CORNER FILL

An optional keyword that directs an Expand Edge operation to fill gaps between rectangles formed by the operation at corners of the input layer. You cannot specify CORNER FILL with a BY FACTOR optional keyword or an EXTEND optional keyword.

## Description

Expands all edges of *layer* into rectangles. Can also extend or retract all edges of *layer* prior to expanding them into rectangles. If extended or retracted, the product of the *factor* parameter and the edge's length is computed after the edge has been extended or retracted.

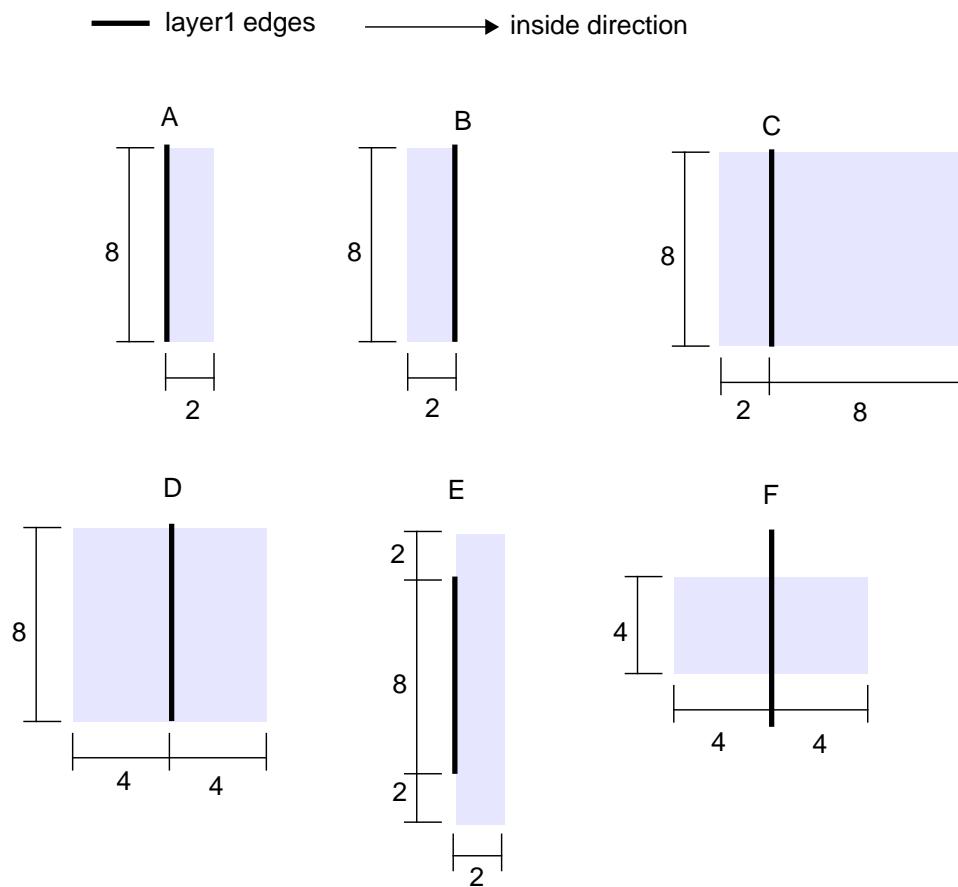
The Expand Edge operation with CORNER FILL specified only fills corners which actually point in the direction of the expansion. For the OUTSIDE BY option, the internal angle at the corner must be less than 180 degrees. For the INSIDE BY option, the internal angle at the corner must be greater than 180 degrees.

Expand Edge can be used in some cases to reduce the layer data that other layer operations have to manipulate. This is done to increase rule file efficiency. See “[Efficient Width and Spacing Checks](#)” in the *Calibre Verification User’s Manual*.

## Examples

Figure 4-93 shows several Expand Edge operations. Each operation expands all *layer* edges into rectangles according to the specified parameters. The operations containing an extension parameter first extend or retract all *layer* edges according to the specified extension parameters and then expand all *layer* edges into rectangles.

Figure 4-93. Expand Edge Operation



A — expand edge layer1 inside by 2

B — expand edge layer1 outside by 2

C — expand edge layer1 outside by 2 inside by factor 1

D — expand edge layer1 by factor 0.5

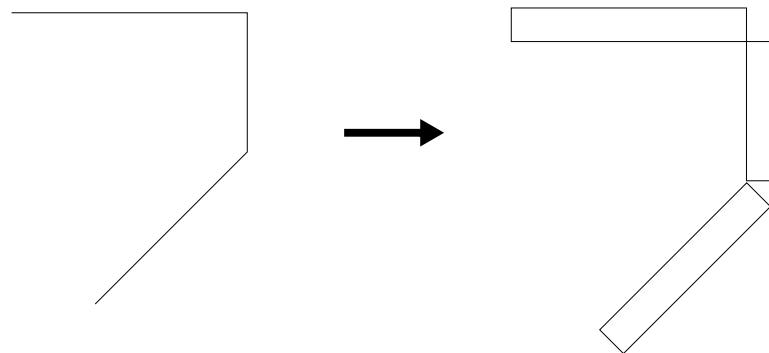
E — expand edge layer1 inside by 2 extend by 2

F — expand edge layer1 by factor 1 extend by factor -0.25

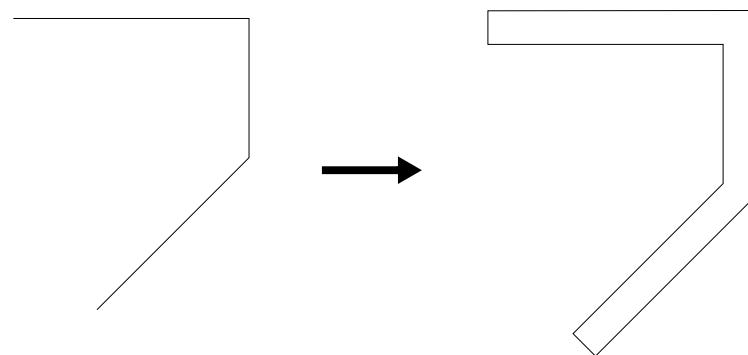
Figure 4-94 illustrates how the CORNER FILL optional keyword operates.

**Figure 4-94. Expand Edge With CORNER FILL**

EXPAND EDGE layer OUTSIDE BY 1



EXPAND EDGE layer OUTSIDE BY 1 CORNER FILL



## Expand Text

Layer operation

**EXPAND TEXT** *text\_name* [*text\_layer*] **BY** *number*  
[PRIMARY ONLY] [CASE SENSITIVE]

### Parameters

- ***text\_name***  
A required name of a text object. Can be a string variable (see [Variable](#)). The ***text\_name*** parameter can contain one or more question mark (?) wildcard characters, where the (?) matches zero or more characters.
- ***text\_layer***  
An optional original layer that contains the ***text\_name***. Used to prevent ambiguity in selecting of text objects having the same name but appearing on different layers.
- **BY *number***  
A required keyword followed by a positive, floating-point number, interpreted in user units, that specifies the size of marker squares.
- **PRIMARY ONLY**  
An optional keyword that specifies only top-cell text is used in the operation.
- **CASE SENSITIVE**  
An optional keyword that specifies ***text\_name*** parameters must match by case in order to be expanded.

### Description

Creates a derived polygon layer consisting of merged squares centered on the positions of text objects having the specified ***text\_name***. The squares have edge length of ***number***.

The parameter order must be observed to avoid ambiguity. The ***text\_name*** is case-insensitive by default.

The Expand Text operation is a non-node-preserving layer constructor.

See also [Expand Cell](#) and [Expand Cell Text](#).

### Examples

```
x = EXPAND TEXT VDD text_layer BY 0.2
//place 0.2 x 0.2 markers on layer x for VDD text locations on text_layer
```

## Extent

Layer operation

**EXTENT** [*layer*]

### Parameters

- *layer*

An optional original layer or layer set, or a derived polygon or edge layer.

### Description

Generates a derived polygon layer that represents the least enclosing rectangle of the database. If you specify the optional *layer* parameter, the rectangle represents the minimum bounding box of all polygons on *layer*. In hierarchical mode, Calibre may choose to divide the rectangle into polygons that are distributed across the hierarchy to facilitate more efficient processing of subsequent operations.

The Calibre database extent consists of layers that are *required* for nmDRC results, connectivity, or device recognition. Layers in the database that are not needed for runtime processing are not read in and do not contribute to the database extent. If you want to ensure that layers not required for the flow contribute to the definition of extent, then see the [Extent Drawn](#) operation.

ICverify views the extent as that of the top-level cell template of the appropriate window.

Calibre expands text basepoints by 2 dbu in all directions. This affects the extent of text layers.

See also [Extent Cell](#) and [Extents](#).

**Examples****Example 1**

Figure 4-95 shows a basic example of an Extent operation. Note how Extent compares to Extents in Figure 4-97.

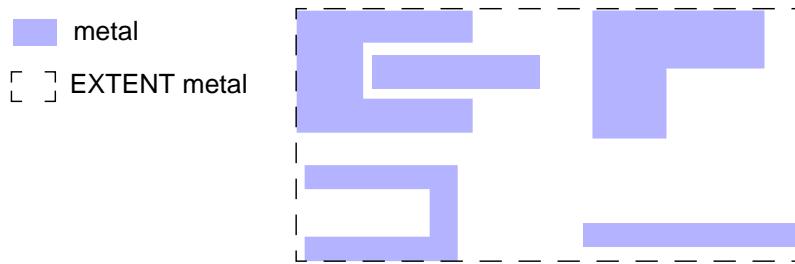
**Figure 4-95. Extent****Example 2**

Figure 4-96 shows a layout with a derived layer called bulk. Assume a simple rule file as follows:

```
Layer nwell 1
Layer poly  2
Layer met   3 // not needed in the run

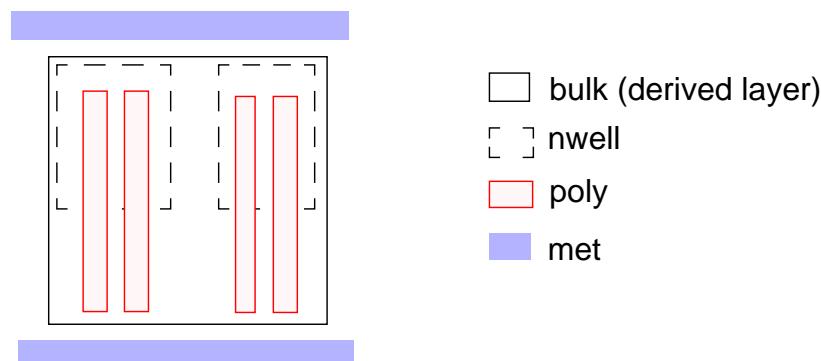
bulk = EXTENT

rule {copy poly copy nwell}
```

Note that layer met plays no role in deriving bulk in this rule file, because met is not needed in a rule check.

If you want all original layers in the database to contribute to the definition of bulk, then you may want to use this derivation instead:

```
bulk = EXTENT DRAWN ORIGINAL
```

**Figure 4-96. Bulk = EXTENT**

## Extent Cell

Layer operation

**EXTENT CELL** *cell\_name* [*cell\_name* ...] [{ORIGINAL | MAPPED} | OCCUPIED]  
[WITH MATCH [GOLDEN] | RULE *rule\_name*] ...]

Used only in Calibre.

### Summary

Generates a derived polygon layer consisting of rectangles that represent the extents of cells in the given list.

### Parameters

- ***cell\_name***

A required name of a cell. You can specify ***cell\_name*** any number of times in one statement. Additionally, any number of “\*” wildcards are allowed and each matches zero or more characters. When using wildcards, enclose ***cell\_name*** in quotation marks (“ ”). The ***cell\_name*** can be a string variable. Cell names are case-sensitive.

- **ORIGINAL**

Optional keyword that specifies all objects (geometry and text) in the input layout database are used to compute the specified cell extent(s).

- **MAPPED**

Optional keyword that specifies all objects in the input layout database that are on simple layers specified by the totality of rule file [Layer](#) specification statements are used to compute the specified cell extent(s).

- **OCCUPIED**

Optional keyword, used with ORIGINAL or MAPPED, that specifies only the cells containing geometry required in the Calibre run (including anywhere in their subhierarchy) have their original extents returned; all other cells are ignored (that is, such cells are handled as if ORIGINAL or MAPPED were not specified).

- **WITH MATCH [GOLDEN] | RULE *rule\_name*] ...**

Optional keyword set that allows a placed cell to be treated as a ***cell\_name*** parameter to the operation if this cell geometrically matches another cell (unplaced) that is already specified as a ***cell\_name*** parameter. The WITH MATCH keyword is particularly useful for matching cells to so-called “golden” cells, when the exact name correspondence may not be known. This keyword set may only be used in hierarchical Calibre applications. The following additional keywords may be specified:

- **GOLDEN**

Causes the operation to attempt to match only cells that appear in designs specified with [Layout Path](#) or [Layout Path2](#) statements that also use the GOLDEN keyword. The GOLDEN keyword may only be specified with the WITH MATCH keyword.

- RULE *rule\_name*

Causes the operation to use [Layout Cell Match Rule](#) statements having a matching *rule\_name* in order to define the cells that are geometrically equal. The Layout Cell Match Rule algorithm is used instead of the default algorithm for determining geometrically equal cells. More than one RULE keyword set may be specified, and this keyword set may only be specified with WITH MATCH. Each *rule\_name* that matches a Layout Cell Match Rule statement is used. Duplications of *rule\_name* are ignored; failure to match a corresponding *rule\_name* causes an LCMR4 compiler error.

## Description

Generates a derived polygon layer consisting of rectangles that represent the extents of cells in the given list.

By default, the cell extents returned by this operation (as well as the database extent returned by the [Extent](#) operation) are the extents of only those objects actually *required* for the Calibre run. For instance, if a layout database contains objects on layer 1 and layer 1 is not required in the Calibre run (that is, it is not required for producing output from the run), then layer 1 is ignored in any Extent Cell calculations of cell extents. (See [Figure 4-97](#) for a related example.)

If ORIGINAL is specified, then all geometry in the input layout database is used to compute cell extents, even if the geometry is not required in the Calibre run. Objects not required for other operations in the Calibre run are subsequently ignored as before. If MAPPED is specified, only mapped layers (those appearing in [Layer](#) statements in the rule file) are used.

If OCCUPIED is specified in addition to ORIGINAL or MAPPED, then the cells having any geometry required in the flow (including anywhere in their sub-hierarchy) have their original extents computed based upon geometry from all layers in the database. All other cells are considered empty, as in the default behavior.

If you use the \* wildcard in a *cell\_name* parameter, the merged extent of all the cells having names that match the search pattern are returned, not the extents of the individual cells.

By default, a warning is issued if a *cell\_name* parameter cannot be found in the layout, but the run proceeds. You can control the severity of this case through the [Layout Input Exception Severity](#) EXTENT\_CELL setting.

The WITH MATCH keyword allows placed cells to be included as part of the Extent Cell calculation based upon geometric matching of another cell used as a *cell\_name* parameter. This feature should only be used with validated cells serving as *cell\_name* parameters in a hierarchical run. Note that specifying “non-golden” cells (cells that have not been previously validated) can result in warnings or errors during the Calibre run.

Before discussing how WITH MATCH works, the term *geometrically equal* is defined. Two cells A and B are said to be *geometrically equal* under the following conditions:

1. Flatten and merge layer1 in A and its sub-hierarchy into the coordinate space of A.
2. Flatten and merge layer1 in B and its sub-hierarchy into the coordinate space of B.

3. Perform a Boolean XOR of the results of Steps 1 and 2. If there are no results of the XOR, then A and B are geometrically equal.

This definition does not include text objects.

The output of WITH MATCH is as follows.

**EXTENT CELL layer1 A<sub>1</sub> A<sub>2</sub> ... A<sub>m</sub> WITH MATCH ...**

where m >= 1, is made equivalent to

**EXTENT CELL layer1 B<sub>1</sub> B<sub>2</sub> ... B<sub>n</sub>**

where n >= 0. Each B<sub>j</sub> is the name of a placed cell (this name may or may not be known) that is geometrically equal to some unplaced cell A whose name matches some A<sub>i</sub> with (possibly) wildcards. The geometry on layer1 that appears in all such cells B<sub>j</sub> is output.

The RULE *rule\_name* keyword causes the matching [Layout Cell Match Rule](#) *rule\_name* statement to be used to determine which cells are geometrically equal.

If the GOLDEN keyword is specified, then the unplaced cells A<sub>i</sub> that the operation attempts to match come from designs specified in Layout Path[2] GOLDEN statements.

See also [Extents](#) and [Extent Drawn](#).

## Examples

### Example 1

```
x = EXTENT CELL cella cellb
// returns the extents of cella and cellb according to layers
// actually needed in the run

y = EXTENT CELL cella cellb ORIGINAL
// returns the extents of cella and cellb according to all
// layers in the database
```

### Example 2

Suppose you have a rule for which the poly spacing is different for the memory and non-memory portions of your design. You want to apply two spacing checks, depending on whether the cell is a memory cell or not. In the file golden.gds, the memory cells appear in the cell bitcel.golden. In layout.gds, the names of the memory cells are not known, so you want to match the memory cells geometrically to bitcell.golden.

```
LAYOUT PATH layout.gds golden.gds // concatenate the layouts.
// layout cell names are not expected to match.
LAYOUT ALLOW DUPLICATE CELL NO // default
// Find the poly in memory cells in layout.gds by geometric matching
// with golden.gds. The cell name bitcel.golden isn't expected to match
// a cell name in layout.gds; however, it is the known memory cell name
// in golden.gds.

poly_in_bitcel = poly AND (EXTENT CELL bitcel.golden WITH MATCH)

40X { @ poly spacing is 0.3, except in bit cells where it is 0.2.

X1 = EXT poly < 0.3 REGION EXTENTS
```

```
// apply 0.3 spacing measurement for the entire design  
  
X2 = EXT poly_in_bitcel < 0.3 REGION EXTENTS  
// apply 0.3 spacing in memory cells;  
// note X1 has already calculated it so there is no runtime penalty.  
  
X3 = X1 NOT X2  
// spacing errors that are not in memory cells  
  
X3 OR ( EXT poly_in_bitcel < 0.2 REGION EXTENTS )  
// apply a different measurement for memory cells and OR with  
// other cells' results  
}
```

Alternatively, you could do this if the golden cells are not to be concatenated with the main design and matched according to layouts that are specified with the GOLDEN keyword:

```
LAYOUT PATH layout.gds  
LAYOUT PATH golden.gds GOLDEN // do not concatenate bitcel.golden with  
// layout.gds  
  
// Find the poly in memory cells in layout.gds by geometric matching  
// with golden.gds.  
  
poly_in_bitcel = poly AND (EXTENT CELL bitcel.golden WITH MATCH GOLDEN)
```

## Extent Drawn

Layer operation

**EXTENT DRAWN** [ORIGINAL | *layer* [*layer* ...]] [IGNORE *layer* [*layer* ...]]

Used only in Calibre.

### Parameters

- **ORIGINAL**  
Optional keyword that specifies all polygons in the input layout database are used to compute extent whether their layers are required by the flow or not. This keyword may not be specified with *layer* parameters immediately following it.
- *layer*  
Optional argument that specifies an original layer name or number. More than one *layer* parameter may be specified.
- **IGNORE *layer***  
Optional keyword that specifies to ignore the indicated *layer* parameter when computing extent. More than one *layer* parameter may be specified.

### Description

Generates a derived polygon layer consisting of the extent of shapes (not text) on the specified input layers.

Without any arguments, Extent Drawn returns the extent of all shapes on all original layers *required* in the flow. That is, layers that are required for results output contribute to the definition of extent. Layers that are not needed by the flow for determining results output are not read in by default. See the [Extent](#) operation for further discussion of this behavior.

If ORIGINAL is specified, then all shapes from each layer in the input layout database are used to compute the extent, even if the shapes are not required in the Calibre run.

If *layer* parameters are specified instead of ORIGINAL, then the shapes on the specified layers are used to compute extent.

If the IGNORE keyword is specified, then the associated *layer* parameters are not used when computing extent.

The benefit of this operation is its efficiency as well as its relative terseness over other approaches. For example, assume we want the extent of original layers m1, m2, ..., m7. The following approach is a bad idea from the standpoint of performance:

```
x = EXTENT ( (m1 OR m2) OR (m3 OR m4) ) OR ( (m5 OR m6) OR m7 ) // Bad.
```

Huge numbers of edges are created by the OR operations, so it is best to avoid this approach. The following example also has serious drawbacks from the standpoint of performance:

```
LAYER MT_LAYER m1 m2 m3 m4 m5 m6 m7
x = EXTENT MT_LAYER // Bad.
```

## Extent Drawn

---

Original layers (MT\_LAYER among them) are merged before use. This method may be worse than the preceding one because the amount of geometry entering the flow may double.

The Extent Drawn operation shown here:

```
x = EXTENT DRAWN m1 m2 m3 m4 m5 m6 m7
```

or here:

```
LAYER MT_LAYER m1 m2 m3 m4 m5 m6 m7  
x = EXTENT DRAWN MT_LAYER
```

computes its result as the input layout database is being read in, and is very fast. In addition, layer parameters not otherwise required in the flow are ignored, as usual, except for their contributions to Extent Drawn operations.

See also [Extent Cell](#) and [Extents](#).

## Examples

### Example 1

```
x1 = EXTENT DRAWN  
// Return the extent of all shapes on all original layers  
// required in the flow.
```

### Example 2

```
x2 = EXTENT DRAWN IGNORE vial via2 62  
// Return the extent of all shapes on all original layers required in the  
// flow, but ignore shapes on layers vial, via2, and 62. (The layers vial,  
// via2, and 62 need only be present in the input layout database and may  
// not necessarily be required in the flow).
```

### Example 3

```
x3 = EXTENT DRAWN ORIGINAL  
// Return the extent of all shapes in the input layout database  
// whether their layers are required by the flow or not.
```

### Example 4

```
x4 = EXTENT DRAWN met1 met2 met3  
// Return the extent of all shapes in the input layout database on layers  
// met1, met2, and met3. (The layers met1, met2, and met3 need only be  
// present in the input layout database and may not necessarily be  
// required in the flow).
```

### Example 5

```
x5 = EXTENT DRAWN ORIGINAL IGNORE vial via2 62  
// Return the extent of all shapes in the input layout database, whether  
// their layers are required in the flow or not, but ignore shapes on  
// layers vial, via2, and 62. (The layers vial, via2, and 62 need only be  
// present in the input layout database and may not necessarily be  
// required in the flow).
```

**Example 6**

```
LAYER met1 10
LAYER met2 20 21 22
LAYER met3 30

x6 = EXTENT DRAWN met1 met2 met3 IGNORE 22
// Return the extent of all shapes in the input layout database on layers
// met1, met2, and met3, but ignore shapes on layer 22.(The layers met1,
// met2, and met3 need only be present in the input layout database and
// may not necessarily be required in the flow. Simple layer 22, which is
// part of met2, does not contribute to layer x6.)
```

## Extents

Layer operation

**EXTENTS** *layer1* [CENTERS [*number*] | SQUARES | INSIDE OF LAYER *layer2*]

### Parameters

- ***layer1***  
An original layer or layer set or a derived polygon layer.
- **CENTERS**  
An optional keyword that generates marker squares at the center of each bounding box instead of the bounding boxes themselves. May not be specified with INSIDE OF LAYER or SQUARES.
- ***number***  
An optional size of the generated marker squares. The marker square has the dimension of *number* × *number* user units. The default value is 1. If the specified *number* is less than 2 dbu, then the *number* is adjusted internally to be 2 dbu.
- **SQUARES**  
An optional keyword that specifies that squares are output rather than minimum bounding boxes. The squares have the same center points as the minimum bounding boxes that are otherwise returned by default. May not be specified with INSIDE OF LAYER or CENTERS.
- **INSIDE OF LAYER *layer2***  
An optional keyword and parameter that specify the extents of *layer1* are taken inside of *layer2*. The *layer2* parameter must specify an original or derived polygon layer. May not be specified with CENTERS or SQUARES.

### Description

Generates a derived polygon layer consisting of the merged minimum bounding boxes of each polygon on *layer1*.

Specifying CENTERS returns marker squares in the centers of the bounding boxes, rather than the bounding boxes themselves. The edge length of the squares is controlled by the *number* parameter.

Specifying SQUARES returns squares instead of minimum bounding boxes. Each square has an edge length equal to the longest edge of the minimum bounding box that would have been returned by default. The center of each square corresponds to the center of the original minimum bounding box.

Specifying INSIDE OF LAYER causes the operation to do the following:

`z = EXTENTS x INSIDE OF LAYER y`

1. For each polygon  $P_i$  ( $i = 1 \dots n$ ) on layer  $y$ , let  $zP_i = \text{EXTENT}(x \text{ INSIDE } P_i)$

That is, for each polygon on layer  $y$ , find the [Extent](#) of layer  $x$  shapes [Inside](#) the layer  $y$  polygon.

2.  $z = zP_1, zP_2, \dots zP_n$  combined and merged

Calibre expands text basepoints by 2 dbu in all directions. This affects the extent of text layers.

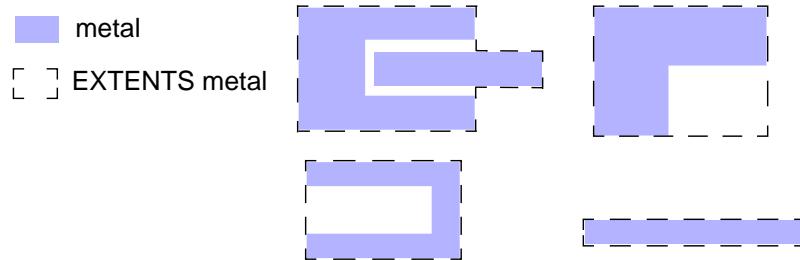
See also [Extent](#) and [Extent Cell](#).

## Examples

### Example 1

[Figure 4-97](#) shows a basic example of an Extents operation. Note how overlapping extents are merged.

**Figure 4-97. Extents**



### Example 2

```
// Center-to-center pad distance must be 300 microns:  
cp = EXTENTS pad CENTERS  
rule { EXT cp < 299 } // Use 299 because centers are 1 x 1
```

## External

Layer operation

Single-layer syntax:

```
EXternal layer1 constraint [metric] [polygon_filter] [connectivity_filter] [orientation_filter]
[bprojection_filter] [angled_filter] [corner_filter] [intersection_filter] [reversal]
[EXCLUDE FALSE] [region_output]
```

Two-layer syntax:

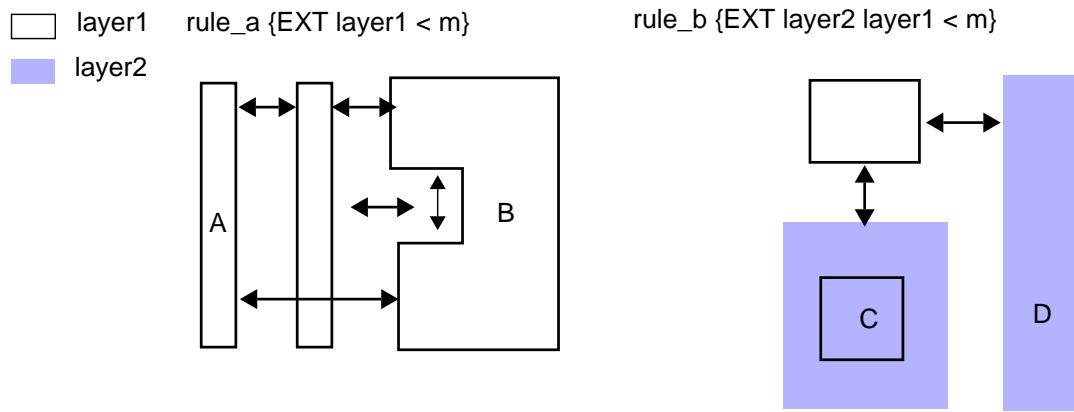
```
EXExternal layer1 layer2 constraint [metric] [polygon_filter] [polygonContainment]
[connectivity_filter] [orientation_filter] [projection_filter] [angled_filter] [corner_filter]
[intersection_filter] [reversal] [EXCLUDE FALSE] [region_output]
```

### Summary

For the single-layer syntax, measures the separations between exterior-facing edges on *layer1*. This includes exterior-facing edges that occur on the same polygon, such as with notches. For the two-layer syntax, measures the separations between the exterior-facing sides of *layer1* edges and the exterior-facing sides of *layer2* edges.

The tool outputs measured edge pairs satisfying the given *constraint*. Intersecting edge pairs and points of singularity are not measured by default. There are multiple optional keyword sets that control the behavior of the EXExternal operation for specialized applications. EXExternal is principally used for spacing checks as shown here:

**Figure 4-98. Basic External Rule Checks**



The rule checks shown in Figure 4-98 measure the separation between the *layer1* edges (single-layer syntax) and *layer1* and *layer2* edges (two-layer syntax). The line segments with arrows indicate where the measurements occur. Notice the order of the layers is unimportant in the two-layer syntax. By default, edge segments from both layers that meet the measurement constraint are output for the two-layer syntax.

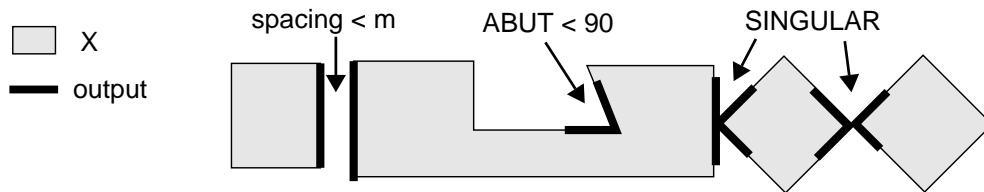
Note that in Figure 4-98, for a sufficiently large value of m, the separation of edges from polygon A to polygon B would also be measured. The intervening polygon between A and B

does not prevent this measurement. However, the separation of edges from polygons C and D is not measured by default because C is contained inside *layer2*.

The default EXternal operation uses the ACUTE ALSO, PARAllel ALSO, NOT PERPendicular, and NOT OBTUSE keywords with the Euclidean measurement metric (which has no keyword).

Figure 4-99 shows the most common type of EXternal rule check. It uses the ABUT and SINGULAR secondary keywords, which are discussed later in this section. ABUT checks intersecting edges and SINGULAR checks single-point interactions. Note that these situations are not checked by default by EXternal.

**Figure 4-99. Typical External Check**



rule\_c {EXT X < m ABUT < 90 SINGULAR}

If EXternal is used to generate derived polygon layers (as opposed to edge layers), it is best to use the REGION EXTENTS keyword, like this:

```
x = EXT lay1 lay2 < 0.04 ABUT < 90 REGION EXTENTS
```

See “[region\\_output](#)” on page 598 for more details about deriving polygon layers with EXternal.

Many fundamental concepts and details of dimensional check operations may be found in the section “Dimensional Check Operations” in the *Calibre Verification User’s Manual*. You may want to have this reference open while reading about External.

Related operations include [Enclosure](#), [Internal](#), [TDDRC](#), [DFM Measure](#), [With Neighbor](#), and [DRC Tolerance Factor](#).

## Parameters and Description

- ***layer1***

An original layer or layer set, or a derived polygon or edge layer.

Enclosing *layer1* in brackets [ ]—called positive edge data—indicates that only edges that meet the ***constraint*** from *layer1* should be output. Enclosing *layer1* in parenthesis ( )—called negative edge data—indicates that edges from *layer1* that do not meet the ***constraint*** should be output. Positive and negative edge data may be used to form derived edge layers. “[Edge-Directed Output](#)” in the *Calibre Verification User’s Manual* for additional details.

- ***layer2***

An original layer or layer set, or a derived polygon or edge layer.

Enclosing ***layer2*** in brackets [ ]—called positive edge data—indicates that only edges that meet the ***constraint*** from ***layer2*** should be output. Enclosing ***layer2*** in parenthesis ( )—called negative edge data—indicates that edges from ***layer2*** that do not meet the ***constraint*** are output. Positive and negative edge data may be used to form derived edge layers. “[Edge-Directed Output](#)” in the *Calibre Verification User’s Manual* for additional details.

Positive (or negative) edge data may be used to derive edge layers as illustrated here:

```
// Poly adjacent to diff < 0.1 um may not be > 5 um in length
diff_poly_ext {
    x = ext [poly] diff < 0.1 ABUT < 90 SINGULAR
    // derived edge layer having edges from poly

    length x > 5 //error output to results database
}
```

Note that derived error layers such as this (notice the [ ] or ( ) operators are not used):

```
x = EXT poly diff < 0.01
```

are usually sent directly to the DRC Results Database and are generally not available for other layer operations to manipulate. However, three DFM operations accept this type of data as input:

- [DFM Analyze](#) — calculates statistics about error data using the COUNT, EC, or EW functions.
- [DFM RDB](#) — writes error layer data to an RDB.
- [DFM Property](#) — filters and/or annotates derived error layers.

- ***constraint***

One of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45, except for  $> a$ ,  $\geq a$ , and  $\neq a$ . Specifies the checking distance, which must have an upper bound. The constraint must contain non-negative real numbers and is interpreted in user units.

---

**Note**

 External operations written to test a greater than or equal zero ( $\geq 0$ ) condition within a range constraint (as in  $\geq 0 \leq 4$ ), do not return a touching or coincident ( $= 0$ ) condition. The ABUT secondary keyword should be used to check  $= 0$  conditions.

---

Using constraints that are very large in comparison to the typical feature size of a layer, and when the layer is rather dense, can result in poor performance. If your performance for an EXternal operation is poor, check to see if the ***constraint*** values are large in comparison to the expected feature sizes of the input layers. If so, attempt to rewrite your rules to reduce the layer data presented to the EXternal operation. See “[Efficient Width and Spacing Checks](#)” in the *Calibre Verification User’s Manual*.

- *metric*

Secondary keyword set that instructs dimensional check operations to use a specified edge measurement metric when measuring the separation between edges. By default, the measurement metric is Euclidean. You do not need to specify any keyword to get a Euclidean measurement.

For detailed information about all of these metrics, refer to the “Metrics” section of the *Calibre Verification User’s Manual*.

There are three basic metrics in addition to the default.

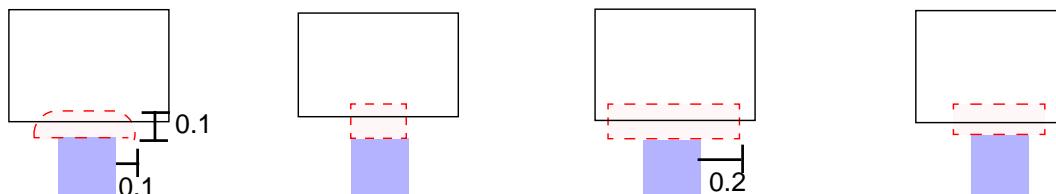
**OPPOSITE** — Specifies extension of the measurement region outward from the edge, but not along the edge, is equal to your **constraint**. It converts the “ $<$ ” constraint to “ $\leq$ ” when measuring intersecting edges. This causes output when intersecting edges abut at 90-degree angles. See the parameter “*intersection\_filter*” on page 591 for a discussion of edge intersection.

**OPPOSITE EXTENDED value** — Specifies to use the opposite metric with an extension of the measurement region along the edge. The *value* is an extension distance in user units and is measured along the edge direction; it must be a positive number.

**SQUARE** — Specifies the extension of the measurement region outward along the edge direction and away from the edge is equal to your **constraint**.

The examples in [Figure 4-100](#) show the four most commonly used measurement regions in red. They show only the measurement regions from ipoly to diff. There are also measurement regions that would be constructed from diff to ipoly.

**Figure 4-100. Measurement Regions**



<pre>ext_ipoly { EXT ipoly diff &lt; 0.1 } // Euclidean</pre>	<pre>ext_ipoly { EXT ipoly diff &lt; 0.1 OPPOSITE }</pre>	<pre>ext_ipoly { EXT ipoly diff &lt; 0.1 OPPOSITE EXTENDED 0.2 }</pre>	<pre>ext_ipoly { EXT ipoly diff &lt; 0.1 SQUARE }</pre>
---	---	--	---

The following are highly specialized metrics used primarily for optical process correction applications. Refer to the “Metrics” section of the [Calibre Verification User’s Manual](#) for complete details.

**OPPOSITE SYMMETRIC** — Specialized metric based upon the OPPOSITE metric. Used for adjusting edge output for non-orthogonal edges.

**OPPOSITE FSYMMETRIC** — Specialized metric based upon the OPPOSITE SYMMETRIC metric. Uses a special fill-in algorithm that outputs single edges, where disjoint edges might otherwise be output.

**OPPOSITE EXTENDED SYMMETRIC *value*** — Specialized metric based upon the OPPOSITE EXTENDED metric. Used for adjusting edge output for non-orthogonal edges.

**OPPOSITE EXTENDED FSYMMETRIC *value*** — Specialized metric based upon the OPPOSITE EXTENDED SYMMETRIC metric. Uses a special fill-in algorithm that outputs single edges, where disjoint edges might otherwise be output.

**OPPOSITE1** — Unidirectional metric based upon the OPPOSITE metric. Similar to OPPOSITE SYMMETRIC but measures edges in one direction from the first input layer to the second input layer.

**OPPOSITE2** — Unidirectional metric based upon the OPPOSITE metric. Similar to OPPOSITE SYMMETRIC but measures edges in one direction from the second input layer to the first input layer.

**OPPOSITE EXTENDED1 *value*** — Unidirectional metric based upon the OPPOSITE EXTENDED metric. Similar to OPPOSITE EXTENDED SYMMETRIC but measures edges in one direction from the first input layer to the second input layer.

**OPPOSITE EXTENDED2 *value*** — Unidirectional metric based upon the OPPOSITE EXTENDED metric. Similar to OPPOSITE EXTENDED SYMMETRIC but measures edges in one direction from the second input layer to the first input layer.

**SQUARE ORTHOGONAL** — Metric used to simulate mask misalignment in the x- and y-directions.

- *polygon\_filter*

The secondary keywords in this set instruct the single-layer EXternal operation to measure the separation between the outsides of edges based upon polygon membership.

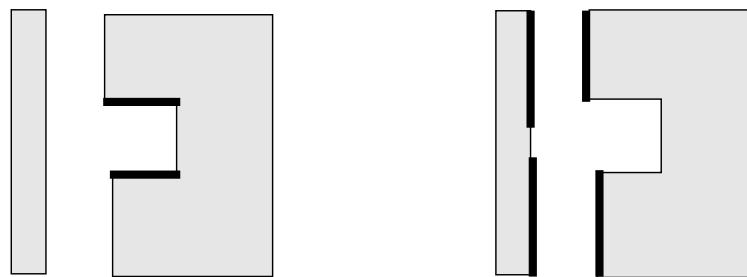
**NOTCH** — Measures the separation between the exterior-facing edges *only* from the same polygon.

**SPACE** — Measures the separation between the exterior-facing edges *only* from different polygons.

If you do not specify either NOTCH or SPACE, then both conditions are measured. Figure 4-101 shows examples.

**Figure 4-101. NOTCH and SPACE**

rule\_d {EXT layer1 < m NOTCH}      rule\_e {EXT layer1 < m SPACE}



NOTCH and SPACE are usually not valid in two-layer dimensional check operations unless both layers have the same layer of origin, as shown here:

```
Rule_85B {
@ Metal to wide metal spacing must be 4 microns. DO NOT check
@ notches. Wide metal is any metal wider than 5 microns.
x = SIZE metal BY -2.5

y = SIZE x BY 2.5 // Wide metal (typical technique).
// Layer z is all edges from the original metal layer which
// are on wide metal. Because z and metal have the same layer
// of origin (metal), we can stipulate to measure spacing.

z = metal COINCIDENT EDGE y // Metal edges on wide metal.
EXT z metal < 4 SPACE // Check spacing only (no notch).
}
```

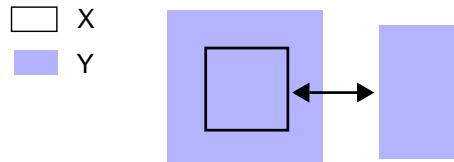
For hierarchical Calibre applications, dimensional check operations with NOTCH and SPACE filters are slower than the identical operations without them; they should be avoided unless needed. For example, it is not necessary to check contact spacing with the SPACE filter if contacts are rectangles by design rule. There are no such considerations in flat applications.

- *polygon\_containment*

The secondary keywords in this set instruct two-layer dimensional check operations to alter the polygon containment criteria when measuring the separation between edges. For more about this topic, refer to the “Polygon Containment Criteria” section of the [Calibre Verification User’s Manual](#).

MEASURE ALL — Specifies to ignore the polygon containment criteria. This allows External to see through polygons that ordinarily it would not. Figure 4-102 shows an example.

**Figure 4-102. MEASURE ALL**



```
rule_f { EXT X Y < m  
        MEASURE ALL  
    }  
// if m is sufficiently large, the  
// indicated measurement is  
// taken
```

- *connectivity\_filter*

Secondary keyword set that instructs dimensional check operations to measure the separation between edges based upon their connectivity. Dimensional check operations ignore connectivity if you do not specify the secondary keywords in this set. Possible choices are:

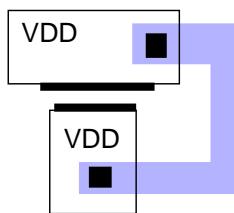
CONNECTED — Specifies to measure only edges from polygons that belong to the same net.

NOT CONNECTED — Specifies to measure only edges from polygons that do *not* belong to the same net.

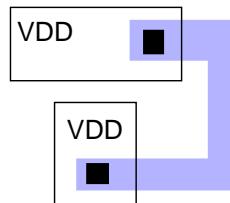
To use these filters, the input *layers* must possess valid connectivity. In [Figure 4-103](#), output occurs based upon connectivity information.

**Figure 4-103. Connectivity-Based Checks**

	X
	Y
	Z



```
connect X Y by Z
rule_e {
  EXT X < m
  CONNECTED
}
```



```
connect X Y by Z
rule_f {
  EXT X < m
  NOT CONNECTED
}
// no output due to
// connect statement
```

For related information, see “Node-Preserving Operations” in the [Calibre Verification User’s Manual](#).

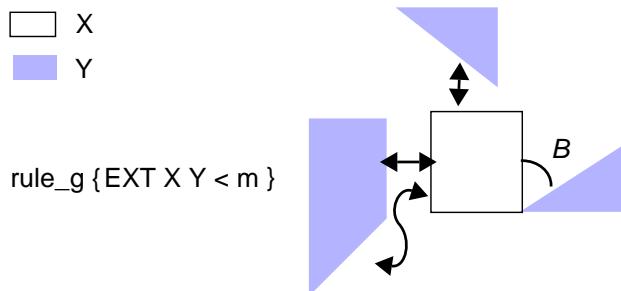
- *orientation\_filter*

Secondary keyword set that instructs the dimensional check operations to measure the separation between edges based upon their appropriate angle or edge orientation. (See the “Appropriateness Criteria” section of the *Calibre Verification User’s Manual*.) You can specify one choice from each of the following four groups in each statement.

The default parameters in this keyword set (PARAllel ALSO, ACUTE ALSO, NOT PERPendicular, NOT OBTUSE) instruct dimensional check operations to measure the separation between the corresponding sides of edges that face each other and that have an appropriate angle of less than 90 degrees.

[Figure 4-104](#) shows the edge orientations that are checked by default using the two-layer EXternal operation for an appropriate value  $m$ . The single-layer form is identical in its behavior. No other orientations are checked unless you select other filters. Note that intersecting (or abutting) edges as shown by angle  $B$  are not checked by default.

**Figure 4-104. Default Edge Orientations Checked by External**



The secondary keywords having ONLY in them cannot be specified with any other secondary keywords from this set. Possible choices are from the following subsets:

*acute\_filter*

**ACUTE ALSO** — Specifies to measure edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees. This is the default behavior if you do not specify a choice from this subset in the operation.

**ACUTE ONLY** — Specifies to measure only edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same EXternal operation.

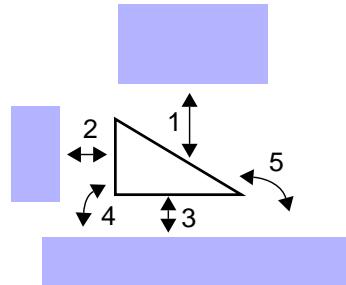
**NOT ACUTE** — Specifies *not* to measure edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees.

Referring to [Figure 4-105](#):

- ACUTE ALSO checks configurations 1 through 3 (this assumes none of the other default settings are changed).
- ACUTE ONLY checks configuration 1.
- NOT ACUTE checks configurations 2 and 3.

Orientations 4 and 5 could be checked if the appropriate filters are set (for example, PERPENDICULAR ALSO and OBTUSE ALSO).

**Figure 4-105. Edge Orientations**



#### *parallel\_filter*

**PARAlleL ALSO** — Specifies to measure parallel edges in addition to non-parallel edges. This is the default behavior if you do not specify a choice from this subset in the operation.

**PARAlleL ONLY** — Specifies to measure only parallel edges. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same EXternal operation.

**NOT PARAlleL** — Specifies *not* to measure parallel edges.

Referring to [Figure 4-105](#):

- **PARALLEL ALSO** checks configurations 1 through 3 (this assumes none of the other default settings are changed).
- **PARALLEL ONLY** checks configurations 2 and 3.
- **NOT PARALLEL** checks configuration 1.

Orientations 4 and 5 could be checked if the appropriate filters are set (for example, PERPENDICULAR ALSO and OBTUSE ALSO).

#### *perpendicular\_filter*

**NOT PERPendicular** — Specifies *not* to measure perpendicular edges. This is the default behavior if you do not specify a choice from this subset in the operation.

**PERPendicular ONLY** — Specifies to measure only perpendicular edges. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same EXternal operation.

**PERPendicular ALSO** — Specifies to measure perpendicular edges in addition to other orientations.

Referring to [Figure 4-105](#):

- **NOT PERPENDICULAR** checks configurations 1 through 3 (this assumes none of the other default settings are changed).

- PERPENDICULAR ONLY checks configuration 4.
- PERPENDICULAR ALSO checks configurations 1 through 4.
- Orientation 5 could be checked if you set OBTUSE ALSO.

*obtuse\_filter*

**NOT OBTUSE** — Specifies *not* to measure edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees. This is the default behavior if you do not specify a choice from this subset in the operation.

**OBTUSE ONLY** — Specifies to measure only edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same dimensional check operation.

**OBTUSE ALSO** — Specifies to measure edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees.

Referring to [Figure 4-105](#):

- NOT OBTUSE checks configurations 1 through 3 (this assumes none of the other default settings are changed).
- OBTUSE ONLY checks configuration 5.
- OBTUSE ALSO checks configurations 1 through 3 and 5.

Orientation 4 could be checked if you set PERPENDICULAR ALSO.

- *projection\_filter*

Secondary keyword set that instructs dimensional check operations to measure the separation between edges based upon their mutual edge projection. Note that the projections of edges onto other edges (if any) are not the output, rather the results of the EXternal operation and associated secondary keywords are output *if the specified projection exists*. For the definition of edge projection, see the discussion of [projection\\_filter](#) in the ENClosure statement.

PROjecting [*projection\_length*] — Specifies to measure the separation between two edges only when one edge projects onto the other edge. If *projection\_length* is also specified (it takes the form of a constraint) and the length of any projection conforms to the *projection\_length* constraint then the results are output. The option's default behavior is *projection\_length* set to  $\geq 0$ .

NOT PROjecting — Specifies to measure the separation between two edges only when neither edge projects onto the other edge.

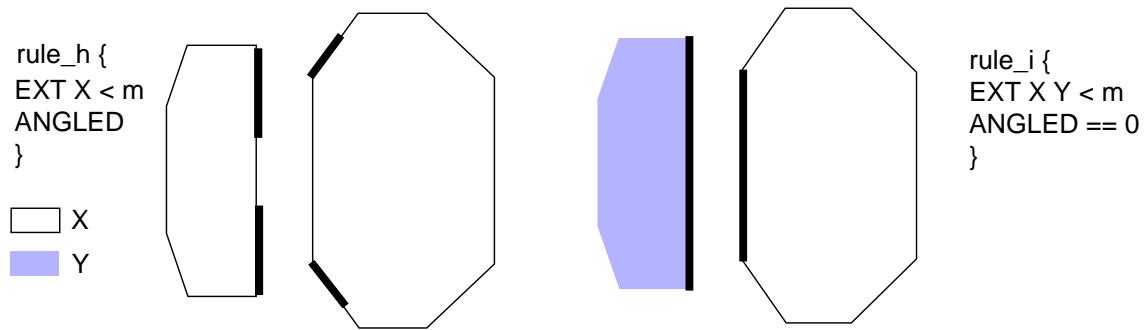
**Hierarchical considerations** — For hierarchical applications, projection and projected length may not be accurately determined if the projection occurs at a hierarchical level well outside the design rule distance between the edges. This problem occurs when two non-parallel edges A and B are being measured with PROJECTING or NOT PROJECTING specified, one or both of the edges extends across multiple hierarchical levels, and the actual projecting portions are well outside the design rule distance between the edges. For constrained PROJECTING filters, if you do not specify the PARALLEL ONLY keyword, Calibre issues a warning and sets the PARALLEL ONLY keyword filter. If you have an unconstrained PROJECTING filter and you want hierarchical and flat results to be the same, it is highly recommended that PARALLEL ONLY be specified.

- *angled\_filter*

Secondary keyword set that instructs dimensional check operations to measure edge pairs based on orthogonality with respect to the coordinate system axes. The possible choice is:

**ANGLED [angle\_constraint]** — Specifies to measure the two edges only when the number of non-orthogonal edges in the pair meets the given constraint. The constraint is optional and when omitted, defaults to > 0. See Figure 4-106.

**Figure 4-106. ANGLED**



### Example

```
EXT poly < 3 ANGLED == 2 // Measure 2 edges only if both are angled.
```

```
EXT poly < 3 ANGLED < 2 // Measure 2 edges only if one or neither is
// angled.
```

- *corner\_filter*

Secondary keyword set that instructs dimensional check operations to measure edge clusters based on corner-to-corner or corner-to-edge orientation.

You cannot specify this filter with an orientation, projection, or angled filter.

**CORNER TO CORNER** [*corner\_constraint*] — Specifies to measure the edge clusters only if they are in a corner-to-corner configuration. The constraint is optional. When you specify *corner\_constraint*, the only allowed options are == 45 and != 45. The constraint limits the measurement of edges according to the angle that the line segment, which links the opposing corners, makes with the x-axis.

**CORNER TO EDGE** — Specifies to measure the edge clusters only if they are in a corner-to-edge configuration (defined later in this section).

**CORNER** — Specifies to measure the edge clusters only if they are in a corner-to-corner or corner-to-edge configuration.

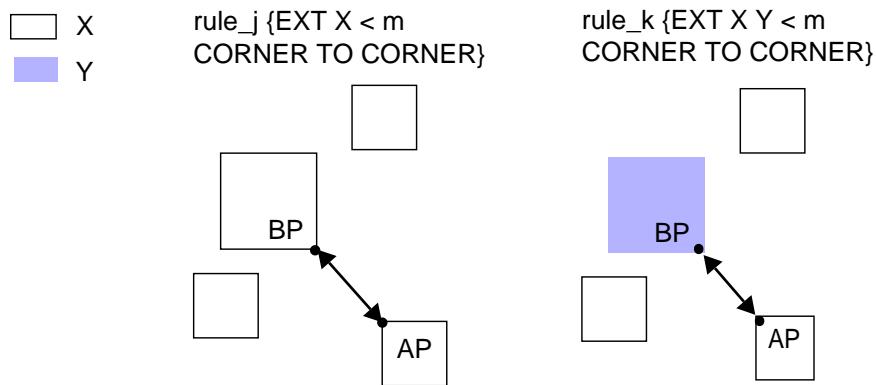
**NOT CORNER** — Specifies to measure the edge clusters only if they are not in a corner-to-corner or corner-to-edge configuration.

The corner-to-corner and corner-to-edge filters are defined as follows: For a single-layer dimension check (input layer X), let A and B be two edges from X to be measured. For a two-layer check (input layers X and Y), let A be the edge from layer X and B the edge from layer Y. A *convex* 90-degree corner of a polygon is one where the polygon inside covers 90 degrees around the corner point and a *concave* 90-degree corner is one where the polygon inside covers 270 degrees.

The edge clusters are in a CORNER TO CORNER configuration when the following hold:

- The edges are parallel and do not project onto each other.
- Let AP be the point on edge A closest to edge B and let BP be the point on edge B closest to edge A. (These points are guaranteed to be unique due to the parallel and projecting restriction.) Both AP and BP lie at convex 90-degree corners of polygons on their respective layers. See Figure 4-107.

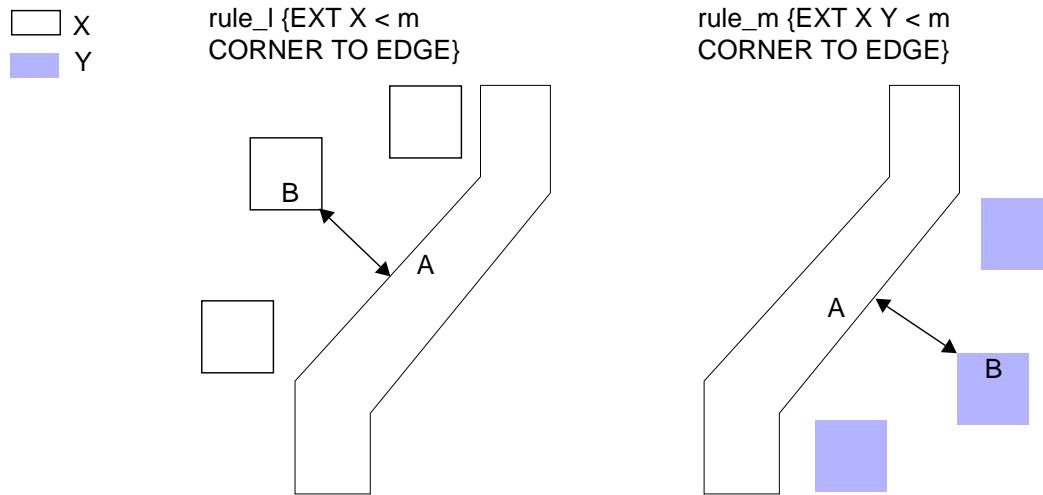
**Figure 4-107. CORNER TO CORNER**



The two edges are in a CORNER TO EDGE configuration when the following hold:

- o The edges are neither parallel nor perpendicular.
- o Either endpoint of edge A or B is at a convex 90-degree corner and the corner projects onto the other edge (that is, the line bisecting the corner intersects the edge at a unique point and this point can be an endpoint of the edge). See Figure 4-108.

**Figure 4-108. CORNER TO EDGE**



### Examples

```

poly_spacing {
  @ poly spacing =2 with the square metric applied for corner-to-edge and
  @ corner-to-corner spacings.
  EXT poly < 2 NOT CORNER //Used to avoid error duplication
  EXT poly < 2 CORNER SQUARE
}
poly_spacing {
  @ poly spacing = 2 with the square metric applied to corner-to-corner
  @ spacings. Corner-to-edge spacing is 2.5
  EXT poly < 2 NOT CORNER //Used to avoid error duplication
  EXT poly < 2 CORNER TO CORNER SQUARE
  EXT poly < 2.5 CORNER TO EDGE
}

```

- *intersection\_filter*

An optional parameter that measures edge pairs based upon intersection behaviors. The value of *intersection\_filter* takes the following form:

[ABUT [*abut\_constraint*]] [OVERLAP] [SINGULAR]

[*intersection\_filter* INTERSECTING ONLY]

You can specify any combination of the secondary keywords ABUT, OVERLAP, and SINGULAR in one operation:

**ABUT [*abut\_constraint*]** — Specifies that separation between intersecting edges should also be checked. It is highly recommended that you use ABUT < 90 in your rule file unless you have good reasons not to. Intersecting (abutting) edges are not checked by default. Intersecting edges are measured *only if* the appropriate angle between them conforms to the optional *abut\_constraint* (interpreted in degrees). With the exception of intersection, the edges must also meet the criteria of the dimensional check operations and any parameters specified from the *connectivity\_filter* or *polygon\_filter* membership sets. Output from the ABUT condition is in addition to any other output the EXTERNAL operation generates.

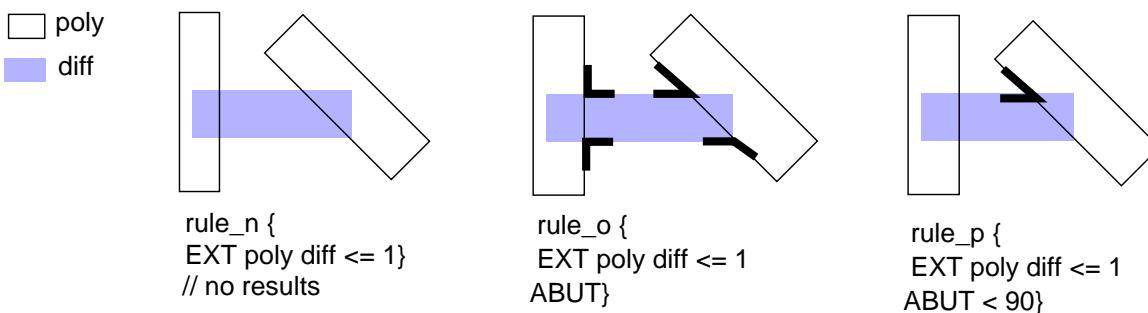
The *abut\_constraint* modifier must contain non-negative real numbers less than 180. Single-operator constraints such as < 90 and > 135 are interpreted as  $\geq 0 < 90$  and  $> 135 < 180$ . The default value is  $\geq 0 < 180$ .

If the *abut\_constraint* modifier includes zero in its range (for example, < 90, == 0,  $\geq 0 < 45$ ), then any edges A of *layer1* and B of *layer2*, which are *coincident outside*, are also output (since the angle between the exterior side of A and the interior side of B is zero). There is no measurement involved in this event and polygon containment criteria are not applied.

In a one-layer dimensional check operation, coincident edges do not exist, since nmDRC applications automatically merge all abutting or overlapping polygons into one polygon. Therefore, the == 0 and the  $\leq 0$  constraint cannot be specified with the ABUT parameter in single-layer syntax.

This measurement of intersecting edges ignores orientation, projection, and corner filters, but does not ignore connectivity, polygon, and angled filters. See Figure 4-109.

**Figure 4-109. ABUT**



**OVERLAP** — Specifies that measurement of the separation between intersecting edges at points where a polygon from one input layer crosses a polygon from the other input layer should also occur. All edges forming the point of overlap are measured as if you specified an unconstrained ABUT parameter (see rule\_o in [Figure 4-109](#)). This overrides any specified ABUT parameter, but only at the point of overlap. You cannot use this parameter when either of the input layers is a derived edge layer because overlaps cannot be computed in the absence of polygons.

The precise definition of a point at which overlap is detected is as follows:

Given a two-layer dimensional check operation between layers A and B, let P be any point where more than one edge from layer A is present and more than one edge from layer B is present (after edge-breaking, see the “Edge Breaking” section of the [Calibre Verification User’s Manual](#) for details). The tool detects an overlap at point P if the following criteria are true:

- An edge from layer A that is outside of or coincident-outside with a polygon from layer B.
- An edge from layer A that is inside of or coincident-inside with a polygon from layer B.
- An edge from layer B that is outside of or coincident-outside with a polygon from layer A.
- An edge from layer B that is inside of or coincident-inside with a polygon from layer A.

**SINGULAR** — Specifies that the separation between the corresponding sides of intersecting edges at points of polygon singularity should also be measured.

*Singularities* are point-to-edge or point-to-point polygon intersections or self-intersections and are often design rule errors. It is highly recommended that you use SINGULAR in your rule file (unless you have good reasons not to) because singularities are not checked by default. Figure [4-110](#) shows examples of singularities detected by an EXTERNAL operation.

When a dimensional check operation detects a point of singularity, all edges forming the point of singularity are measured as if you specified an unconstrained ABUT parameter in the operation. This overrides any specified ABUT parameter but only at the point of singularity.

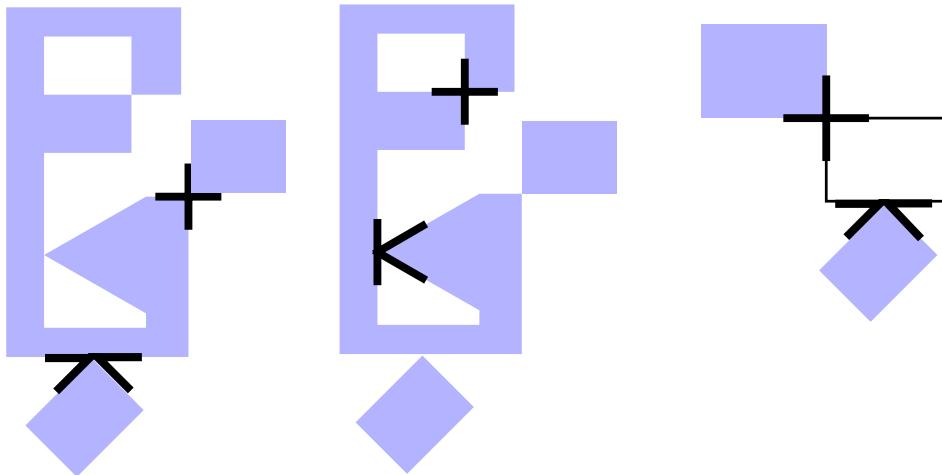
You cannot use the SINGULAR parameter when either of the input layers is a derived edge layer because singularities cannot be computed in the absence of polygons.

**Figure 4-110. External: Measurement and Output at Singularities**

Single-layer  
External SPACE  
SINGULAR

Single-layer  
External NOTCH  
SINGULAR

Two-Layer  
External SINGULAR



**INTERSECTING ONLY** — Specifies to measure only between intersecting edges and to ignore non-intersecting edges. Must be preceded with at least one of the other filters in this set. This filter ignores orientation, angled, projection, and corner filters.

Remember, ABUT, OVERLAP, and SINGULAR output their results *in addition to* the typical EXTERNAL output. INTERSECTING ONLY limits output to only what the *intersection\_filter* keywords produce. Figures 4-109 and 4-110 show the types of output to expect from INTERSECTING ONLY. Figure 4-98 does not because the measured edges do not intersect.

Here are some examples of common configuration of intersection keywords.

### Examples

```
// Poly spacing/notch = 2, exterior acute angles not ok,
// single-point touching not ok.
poly_space { EXT poly < 2 ABUT < 90 SINGULAR }

// Contact spacing = 3, rectangular contacts assumed, single-point
// touching not ok.
cont_space { EXT contact < 3 SINGULAR }

// Poly spacing to diffusion = 1, coincident outside edges not ok
// single-point touching not ok, acute angle crossing not ok.
poly_diff_space { EXT poly diff < 1 ABUT < 90 SINGULAR }

// nwell spacing to ndiff = 3, crossing edges not ok,
// coincident outside edges not ok, single-point touching not ok.
nw_diff_space {
EXT nwell ndiff < 3 ABUT == 0 OVERLAP SINGULAR
}
```

```
// nwell spacing to ndiff = 3, crossing edges ok, coincident
// outside edges ok, single-point touching not ok, acute
// angle crossing not ok.
nw_ndiff_space1 {
EXT nwell ndiff < 3 ABUT > 0 < 90 SINGULAR
}

// nwell spacing to ndiff = 3, crossing edges not ok,
// coincident outside edges ok, single-point touching not ok
// acute angle crossing not ok.
nw_ndiff_space2 {
EXT nwell ndiff < 3 ABUT > 0 < 90 OVERLAP SINGULAR
}
```

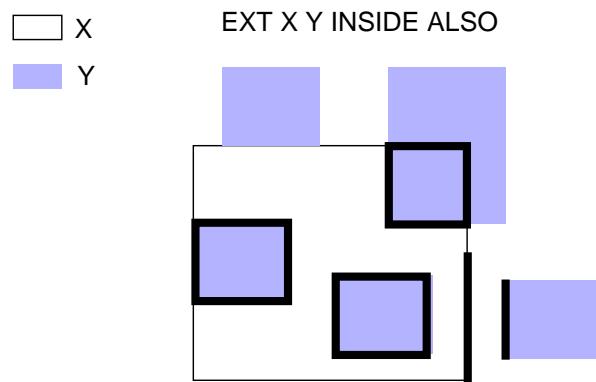
- *reversal*

Secondary keywords that instruct a two-layer EXternal operation to output edges that are enclosed by either *layer1* or *layer2* (Figure 4-111).

**INSIDE ALSO** — Specifies that edges from *layer1* or *layer2*, which are inside or coincident-inside with the other input layer, are output by the operation. Connectivity parameters are obeyed and this is discussed later in this section.

If *layer1* is a derived edge layer, then candidate edges for measurement are restricted to coincident-inside edges from *layer2*. If *layer2* is a derived edge layer, then candidate edges for measurement are restricted to coincident-inside edges from *layer1*.

**Figure 4-111. INSIDE ALSO Output**



The INSIDE ALSO parameter produces one-edge clusters if the output is directed to an error layer; all other error-directed output from a dimensional check operation consists of two-, three-, or four-edge clusters (see “Edge Cluster Generation” in the *Calibre Verification User’s Manual* for details). This behavior is transparent to an edge-directed dimensional check operation and the edge is output according to the layer from which it originated.

Note the INSIDE ALSO parameter generates outputs in the absence of any measurement process by the operation. That is, it is similar to output from an edge-topological operation rather than a dimensional check operation.

The INSIDE ALSO parameter is useful in eliminating time-consuming operations. For example, consider the following design rule:

Contacts and vias must be separated by 3 microns and no stacking or touching is allowed.

Without the INSIDE ALSO parameter, this check requires the following two operations for complete accuracy:

```
contact_via_space {  
    EXTERNAL contact via < 3 ABUT == 0 SINGULAR  
    // Spacing and touching.  
    AND contact via  
    // Common area (stacking).  
}
```

By using the INSIDE ALSO parameter, you can eliminate the AND operation:

**EXTERNAL contact via < 3 ABUT == 0 SINGULAR INSIDE ALSO**

Notes concerning the INSIDE ALSO parameter:

- Because there is no measurement involved in generating output from the INSIDE ALSO parameter, the polygon containment criteria, metrics, and orientation parameters, do not apply.
- The tool obeys connectivity filters for the EXTERNAL operation. If you specify NOT CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it is on the same electrical node. If you specify CONNECTED, then the INSIDE ALSO parameter does not output an edge if the polygon containing it is on a different electrical node. This is helpful in rule checks used for detecting soft connections.
- In the previous examples, spurious errors can be produced. Such errors can be combined by switching to a polygon-directed form (see “[region\\_output](#)” on page 598).
- It is not necessary to use the OVERLAP parameter with INSIDE ALSO if you intend to catch shared or non-shared areas. This is likely to result in spurious errors.

The following is an example where INSIDE ALSO is used with a connectivity filter.

Consider the design rule, followed by the rule check statement:

All n-taps inside nwells must be on the same net as their enclosing nwell.

```
ntap_check {  
    EXTERNAL ntap nwell == 0 INSIDE ALSO NOT CONNECTED  
}
```

The rule check assumes all n-taps are inside an nwell to begin with.

- EXCLUDE FALSE

A secondary keyword used to eliminate rare false errors due to the absence of endpoint blocking edges at the correct hierarchical level when two edges are measured. This is often referred to as a false notch. This only applies to hierarchical runs. This keyword should only be used if false errors are actually detected and they cannot be tolerated. This option has a considerable runtime penalty.

See “[False Measurement Reduction](#)” in the *Calibre Verification User’s Manual* for more details.

- *region\_output*

An optional keyword that instructs dimensional check operations to generate a derived edge or polygon layer instead of a derived error layer. (Derived edge and polygon layers can be further manipulated by other operations, whereas derived error layers cannot.) For detailed information on this subject, see “Polygon-Directed Output” in the *Calibre Verification User’s Manual*.

The value of output takes the following form:

REGION [EXTENTS | CENTERLINE [*value*]]

REGION — Constructs edge projections between the endpoints of selected edges to create polygonal regions; the composite of the selected edges and edge projections is output as derived polygon data.

If you are deriving polygon layers, using the REGION keyword with the Euclidean (default) metric can generate large numbers of skew edges, which must be flattened in hierarchical applications. This is usually wasteful and requires much excess processing time. REGION EXTENTS is usually the best choice for deriving polygon layers. You can also use the OPPOSITE metric to eliminate skew edges, but be aware that you can miss legitimate design rule errors (such as corner-to-corner configurations) when using OPPOSITE.

Skew output edges resulting from the REGION keyword require special handling. The final placement of such edges is the result of merging and numeric rounding. This can result in the placement of skew edge vertices that differ up to 1 dbu from the layout geometry.

REGION EXTENTS — Constructs derived polygon data as for REGION, but the output is the rectangular extents of the polygons output by REGION, rather than the polygons themselves. The tool forms the extents prior to the merging of the regions.

---

**Note**

We recommend using REGION EXTENTS in most layer derivation applications. You should *not* use REGION EXTENTS when polygonal regions are already orthogonal to the database axes, such as those created with the OPPOSITE metric.

---

REGION CENTERLINE [*value*] — Constructs derived polygon data as for REGION. The output consists of the centerlines of the polygonal regions, rather than the regions themselves. These centerlines are formed prior to the merging of the regions. The centerlines are along the direction of the edges whose measurement forms the region; they have a default width of eight database units. The optional parameter *value* allows you to specify the centerline width. The *value* must be a floating-point number greater than or equal to two database units.

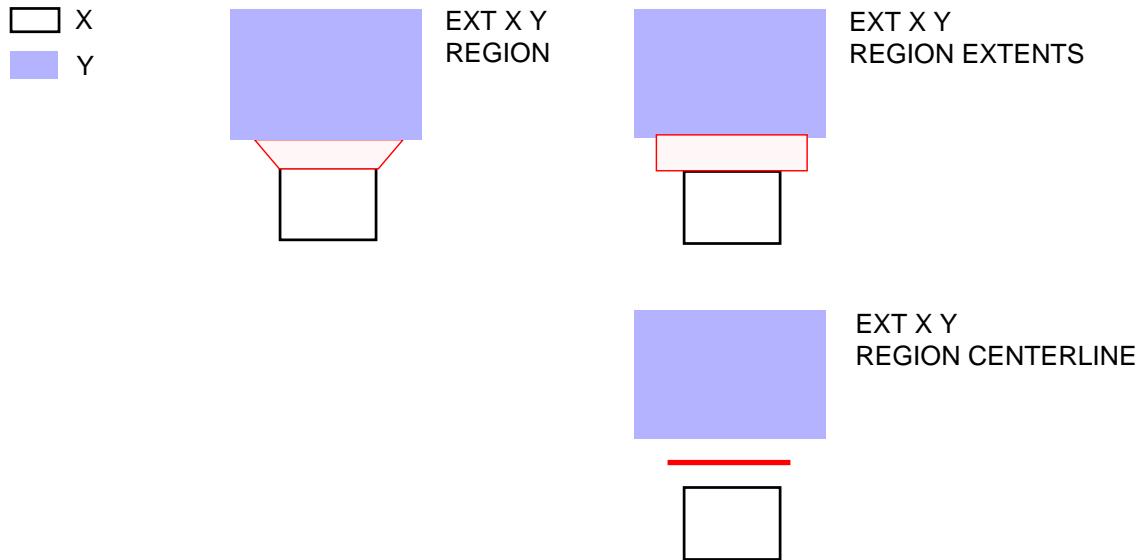
For two, parallel, horizontal edges, the y-coordinate of the centerline segment is always closer to the bottom edge if any snapping occurs. The formula for the y-coordinate of the centerline is:

$$y1 + ( ( y2 - y1 ) / 2 )$$

For vertical edges, a similar change keeps the x-coordinate of the line segment closer to the left edge.

Figure 4-112 shows examples.

**Figure 4-112. REGION Examples**

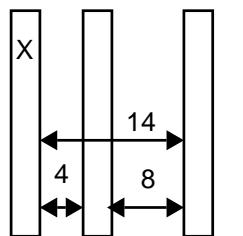


For detailed information on this subject, see “Polygon-Directed Output” in the *Calibre Verification User’s Manual*.

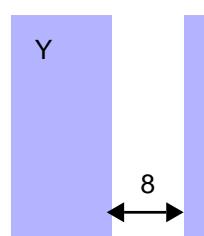
## Examples

### Example 1

If the measurement constraint in an External operation is sufficiently large, the operation can “see through” polygons. Consider the following two sets of operations:



$\text{EXT } X \geq 8 \leq 14$



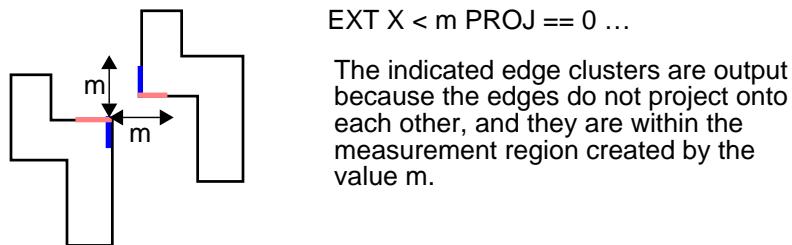
$Y = \text{SIZE } X \text{ BY } 2 \text{ OVERUNDER}$   
 $\text{EXT } Y \geq 8 \leq 14$

The operation on the left takes both the measurements labelled 8 and 14, and selects the appropriate edges, because the constraint interval is large enough for this to occur. For dense line biasing, and similar layer derivations, this is an undesirable behavior.

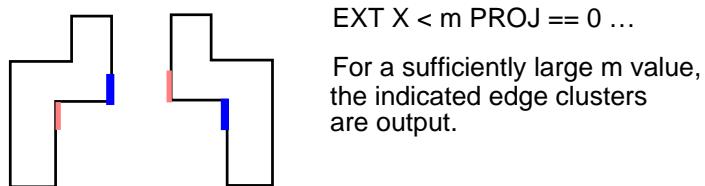
The operation set on the right overcomes this problem. The Size operation causes the original polygons on layer X that are 4 units apart to be merged. The derived layer Y is then presented to the same External operation as before and only the desired measurement is taken.

**Example 2**

The PROJ == 0 setting can be used to check for convex corners that do not project onto each other. This is shown in the following diagram.



However, PROJ == 0 is not limited to simply checking convex corners. It checks any edge pairs that do not project onto each other, and that satisfy the other criteria of the EXT check.



For additional examples see “[Spacing Checks](#)” on page 2076 and “[Contact Checks](#)” on page 2096.

# Flag Acute

Specification statement

**FLAG ACUTE {NO | YES [MAXIMUM WARNINGS {*number* | ALL}]}**

## Parameters

- **NO**

Keyword that instructs the tool *not* to issue a warning when an acute angle is found on an original shape. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to issue a warning when an acute angle is found on an original shape.

- **MAXIMUM WARNINGS {*number* | ALL}**

Optional keyword set specified with **YES** that indicates the number of warnings to report. The default is 100 warnings.

*number* — a positive integer specifying the number of warnings to report.

ALL — specifies that all warnings are reported.

## Description

Controls whether the tool reports a warning whenever an acute angle (less than 90 degrees) is found on an unmerged original shape read from the layout database.

When **YES** is specified, reporting is flat for flat applications and once per cell for hierarchical applications. The reported warning consists of the vertex of each acute angle as well as the layer and cell name of the original geometry. Here is an example of the report:

```
ACUTE angle on layer metal at location (1.79, 1.54) in cell TOP.
```

Unlike the [Drawn Acute](#) operation, the Flag Acute statement does not generate output to the nmDRC results database.

The original layers that are checked are those read during the verification run, not necessarily the entire layer set as referenced in the rule file. For example, if a layer does not appear in a rule check or is not required for connectivity, it is not read in or checked.

You can specify this statement once in a rule file. A maximum of 100 warnings is reported by default.

For additional information, see “[Flagging Bad Original Shapes](#)” in the *Calibre Verification User’s Manual*.

In Pyxis Layout, the default is the value of the Flag Acute statement or the @noflagacute setting of the \$check\_drc() function flagacuteangle switch. If the Flag Acute statement is not present in the rule file, acute angles are not reported unless you set the flagacuteangle switch argument of

## **Flag Acute**

---

the \$check\_drc() function to @flagacute. If you load a new rule file that contains a Flag Acute statement, its setting overrides the previous flagacuteangle switch setting.

See also [Exclude Acute](#).

### **Examples**

```
// Do not flag acute angles in the transcript.  
FLAG ACUTE NO
```

## Flag Angled

Specification statement

**FLAG ANGLED {NO | YES [MAXIMUM WARNINGS {*number* | ALL}]}**

Used only in Calibre.

### Parameters

- **NO**  
Keyword that instructs the tool *not* to issue a warning when an edge that is not orthogonal to the database axes is found on an original shape. This is the default behavior when you do not include this statement in the rule file.
- **YES**  
Keyword that instructs the tool to issue a warning when an edge that is not orthogonal to the database axes is found on an original shape.
- **MAXIMUM WARNINGS {*number* | ALL}**  
Optional keyword set specified with **YES** that indicates the number of warnings to report. The default is 100 warnings.
  - number* — a positive integer specifying the number of warnings to report.
  - ALL — specifies that all warnings are reported.

### Description

Controls whether the tool reports a warning whenever a non-orthogonal edge is found on an unmerged original shape read from the layout database. This includes 45-degree edges, which the [Flag Skew](#) operation does not report.

When **YES** is specified, reporting is flat for flat applications and once per cell for hierarchical applications. A maximum of 100 warnings is reported by default. The reported warning consists of the endpoints of the edge, and both the layer and cell name of the original geometry. Here is an example of the report:

```
ANGLED edge on layer metal from location (1.84, 1.61) to location (1.79,
1.54) in cell TOP.
```

Unlike the [Drawn Angled](#) operation, the Flag Angled statement does not generate output to the nmDRC results database.

The original layers that are checked are those read during the verification run, not necessarily the entire layer set as referenced in the rule file. For example, if a layer does not appear in a rule check or is not required for connectivity, it is not read in and checked.

You can specify this statement once in a rule file.

For additional information, see “[Flagging Bad Original Shapes](#)” in the *Calibre Verification User’s Manual*.

See also [Exclude Angled](#).

## **Flag Angled**

---

### **Examples**

```
// Do not flag angled geometry in the transcript.  
FLAG ANGLED NO
```

## Flag Nonsimple

Specification statement

### FLAG NONSIMPLE [POLYGON] {NO | YES}

Used only in Calibre.

#### Parameters

- **POLYGON**

An optional keyword that can be specified to distinguish from the [Flag Nonsimple Path](#) specification statement. This keyword does not affect the behavior of the specification statement.

- **NO**

Keyword that instructs the tool *not* to issue a warning when a non-simple polygon is found on an original layer. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to issue a warning when a non-simple polygon is found on an original layer.

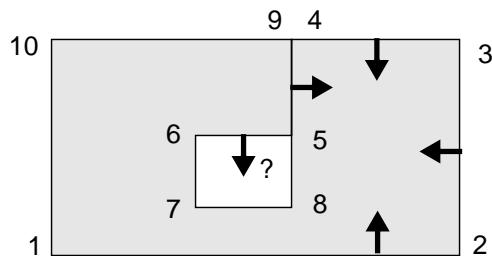
#### Description

Controls non-simple polygon flagging of unmerged original layout database shapes.

Unmerged, original database polygons fall into three categories:

- Non-orientable. These polygons have an inconsistent direction. That is, the inside of the polygon is not consistently to the left or right as the polygon perimeter is traversed. [Figure 4-113](#) shows such a polygon.

**Figure 4-113. Non-Orientable Polygon**



The numerals show a traversal of the polygon vertices, with the polygon interior to the left (indicated by the arrows). From point 5 to point 6, the interior of the polygon becomes inconsistent.

- Non-simple, but orientable. These polygons are orientable but overlap themselves at some point. Overlap means positive area overlap, not outside coincidence of edges.
- Simple. Polygons that are neither of the above.

When **YES** is specified, as the tool reads the database, the tool issues a warning when it encounters and discards a non-orientable polygon. This warning is reported prior to any

## Flag Nonsimple

---

flattening of the input layout database—that is, once in the cell where the polygon is encountered. The tool issues a warning in a similar fashion if a non-simple, but orientable, polygon is encountered. The polygon is not discarded since the overlap is removed when the layer is implicitly merged.

The original layers that are checked are those read during the verification run, not necessarily the entire layer set as referenced in the rule file. For example, if a layer does not appear in a rule check (for nmDRC) or is not required for connectivity, it is not read in and checked.

This statement can be specified only once.

For additional information, see “[Flagging Bad Original Shapes](#)” in the *Calibre Verification User’s Manual*.

See [Layout Input Exception Severity](#) for options in handling non-simple polygons.

## Examples

```
// Do not flag nonsimple polygons in the transcript.  
FLAG NONSIMPLE NO
```

## Flag Nonsimple Path

Specification statement

### FLAG NONSIMPLE PATH {NO | YES}

Used only in Calibre.

#### Parameters

- **NO**

Keyword that instructs the tool *not* to issue a warning when a non-simple path is found as an original shape. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to issue a warning when a non-simple path is found as an original shape.

#### Description

Controls non-simple path flagging of original shapes in geometric layout databases.

Calibre expands paths into polygons upon read in. Unmerged, original database paths fall into two categories:

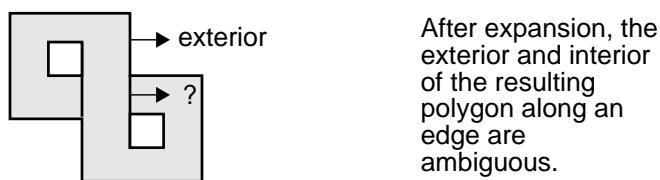
- Non-simple — paths that overlap themselves at some point.
- Simple — paths that do not overlap themselves.

When **YES** is specified, Calibre reports a warning whenever a non-simple path that can be expanded into an orientable polygon is read from the input layout database. A non-simple path is one that results in a non-simple polygon after expansion (see [Flag Nonsimple](#) for the definition of non-simple polygon). The path is still processed since the overlap can be cleanly removed when the resulting polygon is merged. Warnings include a point on the path, its layer, and cell name. Reporting is hierarchical for hierarchical data (that is, once per cell) for geometric layout database formats, and there is no limit on the number of warnings reported.

The [Layout Input Exception Severity](#) PATH\_NONSIMPLE option controls the handling of non-simple paths.

Some non-simple paths can result in non-orientable polygons after expansion. Such paths cause errors by default. The Layout Input Exception Severity PATH\_NONORIENTABLE option controls the handling of such paths. See [Figure 4-114](#).

**Figure 4-114. Non-Simple, Non-Orientable Path Expansion**



## Flag Nonsimple Path

---

The original layers that are checked are those read during the verification run, not necessarily the entire layer set as referenced in the rule file. For example, if a layer does not appear in a rule check, or is not required for connectivity, it is not read in and checked.

This statement can be specified only once.

For additional information, see “[Flagging Bad Original Shapes](#)” in the *Calibre Verification User’s Manual*.

See also [Flag Nonsimple](#).

## Examples

```
// Do not flag nonsimple paths in transcript.  
FLAG NONSIMPLE PATH NO
```

# Flag Offgrid

Specification statement

**FLAG OFFGRID {NO | YES [MAXIMUM WARNINGS {*number* | ALL}]}**

## Parameters

- **NO**  
Keyword that instructs the tool *not* to issue a warning when an off-grid vertex is found on an original shape. This is the default behavior when you do not include this statement in the rule file.
- **YES**  
Keyword that instructs the tool to issue a warning when an off-grid vertex is found on an original shape.
- **MAXIMUM WARNINGS {*number* | ALL}**  
Optional keyword set specified with **YES** that indicates the number of warnings to report. The default is 100 warnings.
  - number* — a positive integer specifying the number of warnings to report.
  - ALL — specifies that all warnings are reported.

## Description

Enables the nmDRC application to report a warning whenever an off-grid vertex is found on an original shape read from the layout database.

When **YES** is specified, reporting is flat for flat applications and once per cell for Calibre nmDRC-H. An off-grid vertex is a point with coordinates that do not follow the layout grid step-size specified in the [Resolution](#) or [Layer Resolution](#) statements. The reported warning consists of the coordinates of the off-grid vertex, the layer, and cell name of the original geometry. The warning is repeated for each off-grid vertex found.

Unlike the [Drawn Offgrid](#) operation, the Flag Offgrid statement does not generate output to the nmDRC results database. To flag off-grid vertices of merged original or derived polygon layers, use the [Offgrid](#) specification statement.

The original layers that are checked are those read during the verification run, not necessarily the entire layer set as referenced in the rule file. For example, if a layer does not appear in a rule check (for nmDRC) or is not required for connectivity, it is not read in and checked.

You can specify this statement once in a rule file. The tool reports a maximum of 100 warnings by default.

For additional information, see “[Flagging Bad Original Shapes](#)” in the *Calibre Verification User’s Manual*.

In Pyxis Layout, the default is the value of the Flag Offgrid statement or the @noflagoffgrid setting of the \$check\_drc() function flagoffgridvertex switch. If the Flag Offgrid statement is not present in the rule file, off-grid vertices are not reported unless you set the flagoffgridvertex

## Flag Offgrid

---

switch argument of the \$check\_drc() function to @flagoffgrid. If you load a new rule file that contains a Flag Offgrid statement, its setting overrides the previous flagoffgridvertex switch setting.

See also [Snap](#), [Snap Offgrid](#), and [Exclude Offgrid](#).

## Examples

```
// Do not flag offgrid vertices in the transcript.  
FLAG OFFGRID NO
```

## Flag Skew

Specification statement

**FLAG SKEW {NO | YES [MAXIMUM WARNINGS {*number* | ALL}]}**

### Parameters

- **NO**

Keyword that instructs the tool *not* to issue a warning when a skew edge is found on an original layer. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to issue a warning when a skew edge is found on an original layer.

- **MAXIMUM WARNINGS {*number* | ALL}**

Optional keyword set specified with **YES** that indicate the number of warnings to report. The default is 100 warnings.

*number* — a positive integer specifying the number of warnings to report.

ALL — specifies that all warnings are reported.

### Description

Enables the nmDRC application to report a warning whenever a skew edge is found on an unmerged original shape read from the layout database. A skew edge is an edge that is not oriented at a multiple of 45 degrees to the database axes. You can specify this statement once in a rule file.

When **YES** is specified, reporting is flat for flat applications and once per cell for Calibre nmDRC-H. The reported warning consists of the endpoints of the skew edge and both the layer and cell name of the original geometry. Here is an example of the report:

```
SKEW edge on layer metal from location (1.84, 1.61) to location (1.79,
1.54) in cell TOP.
```

The tool reports a maximum of 100 warnings by default.

Unlike the [Drawn Skew](#) operation, the Flag Skew statement does not generate output to the nmDRC results database.

The original layers that are checked are those read during the verification run, not necessarily the entire layer set as referenced in the rule file. For example, if a layer does not appear in a rule check (for nmDRC) or is not required for connectivity, it is not read in and checked.

In Pyxis Layout, the default is the value of the Flag Skew statement or the @noflagskew setting of the \$check\_drc() function flagskewedge switch. If the Flag Skew statement is not present in the rule file, skew edges is not reported unless you set the flagaskewedge switch argument of the \$check\_drc() function to @flagskew. If you load a new rule file that contains a Flag Skew statement, its setting overrides the previous flagskewedge switch setting.

## **Flag Skew**

---

See also [Flag Angled](#), [Flag Acute](#), and [Exclude Skew](#).

### **Examples**

```
// Do not flag skew edges in the transcript.  
FLAG SKEW NO
```

# Flatten

Layer operation

## **FLATTEN** *layer*

### Parameters

- *layer*

An original layer or layer set, or a derived polygon or edge layer.

### Description

For Calibre nmDRC-H, the resulting layer is a flat copy of the input layer. For flat Calibre nmDRC and ICrules, this is equivalent to a [Copy](#) operation.

This operation has *limited* use. It can be problematic in that any operation involving a flat layer in Calibre nmDRC-H produces a flat layer. Also, a two-layer operation where exactly one of the input layers is flat must internally flatten the other input layer so that the operation itself can be done flat.

---

#### Note



No flattened temporary layers may be used in any of the DRC functions called from runtime TVF. See [TVF Function](#) and the [TVF layer function](#).

---

See also [Push](#), [Flatten Cell](#), [Flatten Inside Cell](#), and [Merge](#).

### Examples

One use of this operation is to get around the natural error suppression in the hierarchical application. For example:

```
rule6.A { @ Poly spacing to diff = 1u. Show ALL flat errors.
          X = EXT poly diff < 1  ABUT < 90  SINGULAR  REGION
          FLATTEN X
      }
```

Note that polygon-directed output (REGION keyword) is required in the External operation because error-directed output cannot be saved in a derived layer.

## Flatten Cell

Specification statement

**FLATTEN CELL** *cell\_name* [*cell\_name* ...]

Used only in hierarchical Calibre.

### Parameters

- *cell\_name*

A required name of a cell. You can specify *cell\_name* any number of times in one statement. The *cell\_name* can be a string variable (see [Variable](#)). The cell name parameters can contain one or more asterisk (\*) wildcard characters, where the (\*) matches zero or more characters. When using the (\*) character, enclose the cell name in quotes; otherwise, a compilation error occurs because the asterisk is a reserved symbol.

### Description

Specifies the cells whose instances are to be flattened up to the level of the cells in which the instances are placed. That is, the cell and its underlying hierarchy are replaced by a flattened view of the shapes.

Use of this statement is *limited*. In most cases, [Layout Base Layer](#) performs the function of this statement more effectively. Flatten Cell is generally used for standard cell designs.

You use this statement when the design incorporates high-level power rails or bumps as placements instead of geometry. Typically, these placements cover the entire design, which can cause problems in distinguishing the design from a gate array.

You can specify this statement any number of times.

Flatten Cell is not intended to compensate for differences between source and layout hierarchy in nmLVS-H. Proper management of your hcell list avoids any need to use Flatten Cell in such cases. Unnecessary use of Flatten Cell can cause serious performance degradation in nmLVS-H.

See also [Flatten Inside Cell](#) and [Push Cell](#).

### Examples

```
// Flatten these cells one level.  
FLATTEN CELL cella cellb
```

# Flatten Inside Cell

Specification statement

**FLATTEN INSIDE CELL** *cell\_name* [*cell\_name* ...]

Used only in hierarchical Calibre.

## Parameters

- *cell\_name*

A required cell name followed by an optional list of cell names. The *cell\_name* can be a string variable (see [Variable](#)). More than one cell name may be specified. The cell name parameters can contain one or more asterisk (\*) wildcard characters, where the (\*) matches zero or more characters. When using the (\*) character, enclose the cell name in quotes; otherwise, a compilation error occurs because the asterisk is a reserved symbol.

## Description

Specifies that the internal hierarchy of each cell in the parameter list is to be flattened during the hierarchical database construction phase. Applications for this capability are *limited*.

See also [Flatten Cell](#) and [LVS Flatten Inside Cell](#).

## Examples

```
// Flatten the internal hierarchy of these cells.  
FLATTEN INSIDE CELL mux_4  
FLATTEN INSIDE CELL dff_c "dff_c_*
```

# Fracture HITACHI

Mask data prep operation

**FRACTURE HITACHI** *layer* [*layerN...*]  
 [INSIDE OF { *x1 y1 x2 y2* | LAYER *layer\_ext* | EXTENT }]  
 [ FILENAME *output\_file\_name* ]  
 { FILE { *parameter\_block\_name* | '[' *parameter\_list* ']' } }

where *parameter\_list* can contain the following parameters:

```
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}...}]
file_name output_file_name
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[computation_mode {hier | section | hybrid}]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n/d | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[direct_FS_access [INPUT | INTERMEDIATE]...]
[shot_size s]
format_type { 1 | 3 | 7 }
max_skew_approximation_error skew_error //Required only for format_type {1 | 3}
lsb1 number //Required only for format_type 7
[lsb2 number]
[lsb3 number]
[small_value value]
[cd {{internal {auto | cd_max_width cd_min_length} } | {external cd_min_width
  cd_max_width cd_min_length}}]
[section_count_limit limit]
[ssf_size size]
[ssf_margin size]
[comment text]
[large_output_file]
```

Product license: Calibre FRACTUREh

## Parameters

Refer to the “FRACTURE HITACHI (FRACTUREh)” section of the [\*Calibre Mask Data Preparation User’s Manual\*](#) for a complete description of the parameters for this operation.

## Description

Generates a single HITACHI file. For details on mask data preparation, see the [\*Calibre Mask Data Preparation User’s Manual\*](#).

See also [Fracture JEOL](#), [Fracture MEBES](#), [Fracture NUFLARE](#), [Fracture MICRONIC](#), [Fracture VBOASIS](#), and [MDPverify](#).

## Examples

```
frac { FRACTURE HITACHI poly
    INSIDE OF EXTENT
    FILE [
        format 3
        MIRROR x
        ROTATE 90
        MAGNIFY 4
        file_name poly.pfh3
    ]}
```

## Fracture JEOL

Mask data prep operation

**FRACTURE JEOL** *layer* [*layerN...*]  
[INSIDE OF { *x1 y1 x2 y2* | LAYER *layer\_ext* | EXTENT }]  
[ FILENAME *output\_file\_name* ]  
{ FILE { *parameter\_block\_name* | '[' *parameter\_list* ']' } }

where *parameter\_list* can contain the following parameters:

```
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}}...}]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
file_name output_file_name
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[computation_mode {hier | section | hybrid}]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n/d | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[direct_FS_access [INPUT | INTERMEDIATE]...]
[shot_size s]
[device device_name]
[version version_number]
fracture_units gridsize
[fracture_units_2 units]
[exposed_area]
[small_value value]
[cd {{internal {auto | cd_max_width cd_min_length}} | {external cd_min_width
  cd_max_width cd_min_length}}]
[section_count_limit limit]
[{ field_size size /
  {field_size_x x_extent
    field_size_y y_extent}
  }]
[field_oversize extension]
```

[lb\_limit *limit\_value*]  
[hsc\_limit *command\_count\_limit*]  
Product license: Calibre FRACTUREj

## Parameters

Refer to the “FRACTURE JEOL (FRACTUREj)” section of the [Calibre Mask Data Preparation User’s Manual](#) for a complete description of the parameters for this operation.

## Description

Generates a single JEOL file. For details on mask data preparation, see the [Calibre Mask Data Preparation User’s Manual](#).

See also [Fracture HITACHI](#), [Fracture MEBES](#), [Fracture NUFLARE](#), [Fracture MICRONIC](#), [Fracture VBOASIS](#), and [MDPverify](#).

## Examples

```
frac { FRACTURE JEOL poly
    INSIDE_OF_EXTENT
    FILE [
        fracture_units 500
        MIRROR x
        ROTATE 90
        MAGNIFY 4
        file_name poly.jeol
    ]}
```

# Fracture MEBES

Mask data prep operation

**FRACTURE MEBES** *layer* [*layerN...*]  
 [INSIDE OF { *x1 y1 x2 y2* | LAYER *layer\_ext* | EXTENT }]  
 [ FILENAME *output\_file\_name* ]  
 { FILE { *parameter\_block\_name* | '[' *parameter\_list* ']' } }

where *parameter\_list* can contain the following parameters:

```
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}...}]
file_name output_file_name
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[computation_mode {hier | section}]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n/d | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[direct_FS_access [INPUT | INTERMEDIATE]...]
mode {4 | 5}
[auto_mode_switch]
address_size unit_size
[compaction_stripes {1 | 32 | 64}]
[mask_info text]
[angle_fix_yes]
[disk_file_name dfname]
[generate_index_file]
[density]
```

Product license: Calibre FRACTUREm

## Parameters

Refer to the “FRACTURE MEBES (FRACTUREm)” section of the *Calibre Mask Data Preparation User’s Manual* for a complete description of the parameters for this operation.

## Description

Generates a single compacted MEBES file. For details on mask data preparation, see the [Calibre Mask Data Preparation User's Manual](#).

See also [Fracture HITACHI](#), [Fracture JEOL](#), [Fracture NUFLARE](#), [Fracture MICRONIC](#), [Fracture VBOASIS](#), and [MDPverify](#).

## Examples

```
frac { FRACTURE MEBES poly
    INSIDE_OF_EXTENT
    FILE [
        MODE 5
        ROTATE 90
        MIRROR x
        MAGNIFY 4
        address_size 0.001 //microns
        file_name abcdefghi.kl
    ]}
```

# Fracture MICRONIC

Mask data prep operation

**FRACTURE MICRONIC** *layer* [*layerN...*]  
 [INSIDE OF { *x1 y1 x2 y2* | LAYER *layer\_ext* | EXTENT }]  
 [ FILENAME *output\_file\_name* ]  
 { FILE { *parameter\_block\_name* | '[' *parameter\_list* ']' } }

where *parameter\_list* can contain the following parameters:

```
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}...}]
file_name output_file_name
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[computation_mode {hier | section | hybrid}]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n/d | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[direct_FS_access [INPUT | INTERMEDIATE]...]
[shot_size s]
[small_value 0.1 | value]
[cd {{internal {auto | cd_max_width cd_min_length}} |
  {external cd_min_width cd_max_width cd_min_length}}]
[section_count_limit limit]
version {1.6 | 1.8}
[scale [+integer | +float | +int_numer +int_denom]]
[shift_origin_ll {0 | 1}]
[runtime_exception_severity file_name_restrictions [-1 | 0 | 1]]
[comment text]
[enable_checksum]
[density wx wy file_name]
```

Product license: Calibre FRACTUREc

## Parameters

Refer to the “FRACTURE MICRONIC (FRACTUREc)” section of the *Calibre Mask Data Preparation User’s Manual* for a complete description of the parameters for this operation.

## Description

Generates a single compacted Micronic file. For details on mask data preparation, see the *Calibre Mask Data Preparation User’s Manual*.

See also [Fracture HITACHI](#), [Fracture JEOL](#), [Fracture MEBES](#), [Fracture NUFLARE](#), [Fracture VBOASIS](#), and [MDPverify](#).

## Examples

```
MY_FRACTURE { FRACTURE MICRONIC POLY FILE [
    version 1.6
    file_name "mdp/m8051.mic"
    log_file_name reports/mic.log
]
}

// bad = MDPVERIFY POLY FILE [
//     input_file mdp/m8051.mic
//     verify_type MICRONIC2DB
// ]

// bad { COPY bad } DRC CHECK MAP bad 300
]
```

## Fracture NUFLARE

Mask data prep operation

**FRACTURE NUFLARE** *layer* [*layerN...*]  
[INSIDE OF { *x1 y1 x2 y2* | LAYER *layer\_ext* | EXTENT }]  
[ FILENAME *output\_file\_name* ]  
{ FILE { *parameter\_block\_name* | '[' *parameter\_list* ']' } }

where *parameter\_list* can contain the following parameters:

```
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}...}]
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[computation_mode {hier | section | hybrid}]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n[/d] | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[direct_FS_access [INPUT | INTERMEDIATE]...]
[shot_size s]
```

The following are VSB11 options:

```
version 11
chip_directory output_directory
max_skew_approximation_error skew_error
[conversion_address_unit {0.00125 | unit}]
[frame_width {512 | width}]
[cell_max_height {1024 | mheight}]
[cell_max_width {128 | mwidht}]
[cell_subfield_size {64 | size}]
[cell_subfield_margin {0.25 | size}]
[frame_max_data {256 | fdata}]
[common_max_data {256 | cdata}]
[frame_bde_max_data {256 | fbdedata}]
```

---

```
[common_bde_max_data {256 | cbdedata}]
[runtime_exception_severity bde_filesize_limit [0 | 1]]
[runtime_exception_severity frame_width_range [0 | 1]]
[pattern_array_compression {0 | 1}]
[pattern_compression {0 | 1}]
[pattern_set description]
[chip_name name]
[array_cell_cost_multiplier {4 | multiplier}]
```

The following are VSB12 options:

```
version 12
chip_directory output_directory
max_skew_approximation_error skew_error
[block_width {1024 | value}]
[block_height {128 | value}]
[frame_width {512 | width}]
[subframe_width {8 | value}]
[cell_max_height {128 | mheight}]
[cell_max_width {128 | mwidth}]
[frame_max_num_cell_locations {4000000 | num}]
[chip_name name]
[frame_max_data {256 | fdata}]
[common_max_data {256 | cdata}]
[conversion_address_unit {0.001 | unit}]
[cell_subfield_size {128 | size}]
[cell_subfield_margin {0.5 | size}]
[pattern_composite_rep {0 | 1}]
[pattern_array_compression {0 | 1}]
[pattern_set description]
[pattern_compression {0 | 1}]
[runtime_exception_severity frame_width_range [0 | 1]]
[array_cell_cost_multiplier {4 | multiplier}]
[remote_direct_merge {0 | 1}]
[device {ebm6000 | ebm7000}]
```

Product license: Calibre FRACTUREt

## Parameters

Refer to the “FRACTURE NUFLARE (FRACTUREt)” section of the [Calibre Mask Data Preparation User’s Manual](#) for a complete description of the parameters for this operation.

## Description

Generates a single compacted VSB11 or VSB12 file. For details on mask data preparation, see the [Calibre Mask Data Preparation User’s Manual](#).

See also [Fracture HITACHI](#), [Fracture JEOL](#), [Fracture MEBES](#), [Fracture MICRONIC](#), [Fracture VBOASIS](#), and [MDPverify](#).

### Examples

```
// VSB11 fracture of poly

frac { FRACTURE NUFLARE poly
    INSIDE_OF_EXTENT
    FILE [
        chip_directory "./fracture_design"
        version 11
        max_skew_approximation_error 0.005
        ROTATE 90
        MIRROR x
        MAGNIFY 4
        chip_name mem_05u
    ]}
```

# Fracture VBOASIS

Mask data prep operation

**FRACTURE VBOASIS** *layer* [*layerN...*]  
 [INSIDE OF { *x1 y1 x2 y2* | LAYER *layer\_ext* | EXTENT }]  
 [ FILENAME *output\_file\_name* ]  
 { FILE { *parameter\_block\_name* | '[' *parameter\_list* ']' } }

where *parameter\_list* can contain the following parameters:

```
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}...}]
file_name output_file_name
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[computation_mode {hier | section | hybrid}]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n/d} | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[direct_FS_access [INPUT | INTERMEDIATE]...]
[shot_size s]
fracture_units gridsize
[small_value value]
[cd { {internal {auto | cd_max_width cd_min_length}} | {external cd_min_width
  cd_max_width cd_min_length}}}]
[section_count_limit limit]
[cell_size cs]
[cell_size_x cx]
[cell_size_y cy]
[depth d]
[oasis_vsb_conformance_severity {0 | 1 | 2}]
[positive_output_extent]
[cblock]
[exposed_area]
```

Product license: Calibre FRACTUREv

## Parameters

Refer to the “FRACTURE VBOASIS (FRACTUREv)” section of the [\*Calibre Mask Data Preparation User’s Manual\*](#) for a complete description of the parameters for this operation.

## Description

Generates a VB:OASIS (Vector-Beam: Open Artwork System Interchange Standard) output file. For details on mask data preparation, see the [\*Calibre Mask Data Preparation User’s Manual\*](#).

See also [Fracture HITACHI](#), [Fracture MEBES](#), [Fracture NUFLARE](#), [Fracture MICRONIC](#), and [MDPverify](#).

## Examples

```
frac1 { FRACTURE VBOASIS hole INSIDE OF EXTENT FILE [
    fracture_units 1000
    magnify 4/1
    file_name TEST1XXXX.VBO
    log_file_name  TEST1XXXX.VBO.log
]}
```

## Fracture OASIS\_MASK

Mask data prep operation

```
FRACTURE OASIS_MASK layer [layerN...]
  [INSIDE OF { x1 y1 x2 y2 | LAYER layer_ext | EXTENT}]
  [ FILENAME output_file_name ]
  { FILE { parameter_block_name | '[' parameter_list ']' } }
```

where *parameter\_list* can contain the following parameters:

```
[<SVRFSTART>
  svrf_statements
<SVRFEND>]
[svrf_range r]
[embedded_svrf_scale {WAFER | MASK}]
[svrf_layer_name {layer_name | {layer_name oasis_layer_number
  [oasis_layer_datatype]}}...}]
file_name output_file_name
[input_region_severity {0 | 1}]
[index_options [-topcell cellname]]
[lint]
[log_file_name name [-dump]]
[mirror {x | y | x y | y x}]
[rotate {0 | 90 | 180 | 270}]
[magnify mag]
[reverse_tone]
[computation_mode {hier | section | hybrid}]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n/d} | AUTO}]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[direct_FS_access [INPUT | INTERMEDIATE]...]
[shot_size s]
fracture_units gridsize
[cell_size cs]
[cell_size_x cx]
[cell_size_y cy]
[depth d]
[oasis_vsb_conformance_severity {0 | 1 | 2}]
[positive_output_extent]
[cblock]
[exposed_area]
[speed_mode]
```

Product license: Calibre FRACTUREv

## Parameters

Refer to the “FRACTURE OASIS\_MASK (FRACTUREv)” section of the [\*Calibre Mask Data Preparation User’s Manual\*](#) for a complete description of the parameters for this operation.

## Description

Generates an OASIS\_MASK format file. OASIS\_MASK is the MDP file format conforming to the SEMI P44-0708 standard. Cell localization and all the properties associated with it are not supported. OASIS\_MASK is a restriction of the OASIS format and a superset of the VB:OASIS format. Files generated by FRACTURE OASIS\_MASK can be used wherever OASIS and VB:OASIS files can be used.

For details on mask data preparation, see the [\*Calibre Mask Data Preparation User’s Manual\*](#).

See also [Fracture HITACHI](#), [Fracture MEBES](#), [Fracture NUFLARE](#), [Fracture MICRONIC](#), [Fracture VBOASIS](#) and [MDPverify](#).

## Examples

```
frac1 { FRACTURE OASIS_MASK hole INSIDE OF EXTENT FILE [
    fracture_units 1000
    magnify 4/1
    file_name TEST1XXXX.VBO
    log_file_name TEST1XXXX.VBO.log
]}
```

## Group

Specification statement

**GROUP** *name rule\_check [rule\_check ...]*

### Parameters

- ***name***

A required name of a rule check group. This name cannot match the name of a rule check or another rule check group in the rule file.

- ***rule\_check***

A required rule check name or group name. You can specify this parameter any number of times for each statement. A ***rule\_check*** can contain one or more question mark (?) characters; the (?) character is a wildcard that matches zero or more characters.

### Description

Names a collection of rule check statements. This statement lets you predefine (within the rule file) sets of rule check statements that are executed together so you do not have to re-specify them each time you want to run that unit. Also, predefining rule check statements gives you a way to express intrinsic relationships between various design rules.

Wildcard matching is only applied to rule check names in the group definition; rule check *group* names referenced in the definition must match exactly.

Selecting and executing a rule check group with [DRC Select Check](#) is equivalent to selecting and executing all rule check statements defined in the group. You can have unlimited hierarchy in a rule check group definition. Rule check group names do not appear in the nmDRC results database; they only serve as a means of predefining sets of rule checks for combined selection and execution.

See also [DRC Unselect Check](#), [ERC Select Check](#), [ERC Unselect Check](#).

### Examples

```

GROUP metal_width met?width
GROUP metal_spacing met?space
GROUP poly_metal poly_metal_?
GROUP contact_metal contact_metal_?
...
//Put metal checks together in one group
GROUP metal_checks metal_width metal_spacing poly_metal contact_metal

DRC SELECT CHECK metal_checks //Do just the metal checks

```

## Grow

Layer operation

**GROW** *layer* [RIGHT BY *value*] [TOP BY *value*] [LEFT BY *value*] [BOTTOM BY *value*]  
[SEQUENTIAL]

### Parameters

- *layer*  
A required original or derived polygon layer, or derived edge layer.
- *value*  
A non-negative floating-point number in user units. If SEQUENTIAL is specified, each *value* may also be negative.

At least one of the following four classifications must be specified:

- RIGHT BY  
An optional keyword set that instructs the tool to expand the *right* edge(s) of objects on the input *layer* by the specified *value*.
- TOP BY  
An optional keyword set that instructs the tool to expand the *top* edge(s) of objects on the input *layer* by the specified *value*.
- LEFT BY  
An optional keyword set that instructs the tool to expand the *left* edge(s) of objects on the input *layer* by the specified *value*.
- BOTTOM BY  
An optional keyword set that instructs the tool to expand the *bottom* edge(s) of objects on the input *layer* by the specified *value*.
- SEQUENTIAL  
An optional keyword that specifies that the Grow operation is performed iteratively using the order of the RIGHT BY, TOP BY, LEFT BY, and BOTTOM BY parameters as they appear in the operation from left to right.

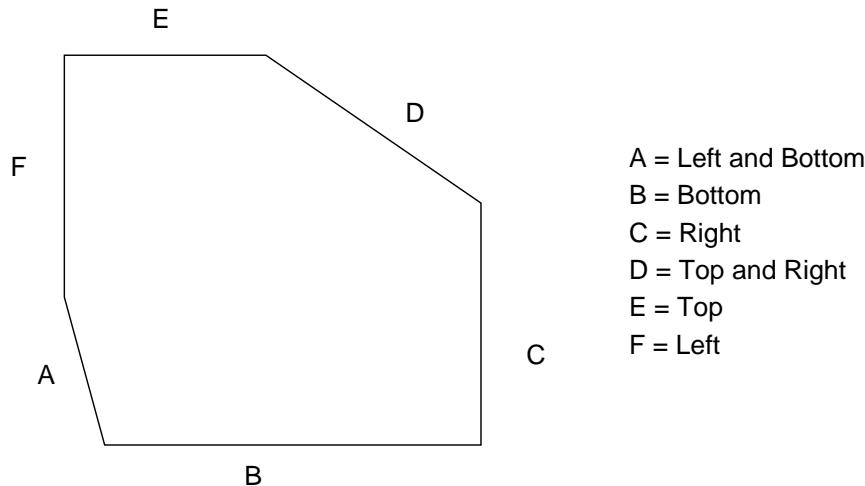
### Description

Allows objects on the input layer to be expanded in the direction of the x-axis, y-axis, or both.

To determine which edges are expanded, the tool uses the following:

All orthogonal edges (with respect to the database axes) on the input layer are classified as right, top, left, or bottom edges dependent on how the *outside* of the edge faces. Non-orthogonal edges receive two of the four edge classifications. Figure 4-115 shows how six sides of a polygon are classified by the Grow operation.

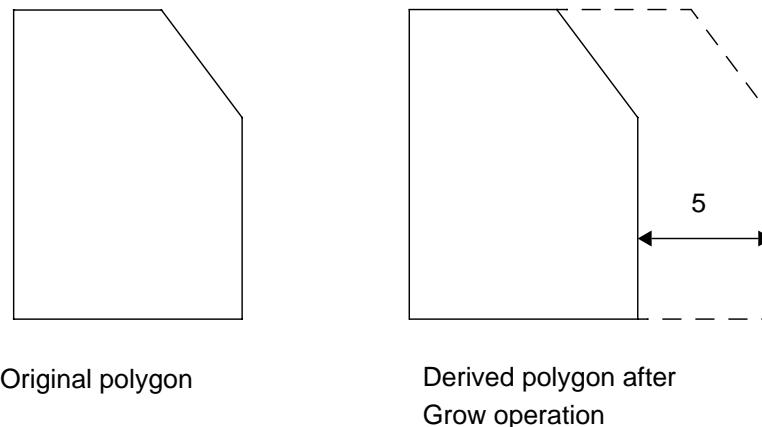
**Figure 4-115. Edge Classification of the Grow Operation**



After the edges on the input layer have been classified the tool then expands edges based on their classification and which optional keyword sets you specified in the rule file statement. The amount of expansion is taken from the *value* parameter assigned to each optional keyword set. Figure 4-116 shows an example of a Grow operation using the RIGHT BY optional keyword set.

**Figure 4-116. Grow Operation Example**

Grow layerA RIGHT BY 5



When *layer* is a derived edge layer, all of the resulting expansion is merged and output. Otherwise, all of the resulting expansion plus the input layer are merged together and output.

For hierarchical applications, degradation of the output layer and performance of the operation can occur due to cells that do not have consistent transforms to the flat view of the design and as a function of the parameters to the operation. Geometry must be recursively promoted until consistency can be achieved, the objects can then be grown in the cell where the consistency is

achieved. This insures that results are correct from the flat viewpoint of the design. For example, the operation:

**GROW layerX RIGHT BY 1 LEFT BY 1**

cannot be done within any cell where there is a flat view transform of a placement of the cell having a 90 or 270 degree rotational component and a flat view transform which does not. For instance, if placements A and B exist in cell C, you cannot do a Grow operation on a layer in C if placement B is rotated 90 or 270 degrees with respect to A. This is due to the directions TOP, RIGHT, BOTTOM, and LEFT being different in placements A and B.

If SEQUENTIAL is specified, *layer* must be an original or derived polygon layer (it cannot be a derived edge layer). Each *value* must be a floating point number in user units and may be negative or positive. Each edge that makes up *layer* is classified according to any number of the specified keywords RIGHT BY, TOP BY, LEFT BY, and BOTTOM BY (these can appear in any order). Edges of *layer* belonging to each classification are expanded and recombined with *layer* in the following sequence:

Let L = *layer*

For each keyword set classification K in order from left to right:

Let X = edges in L belonging to K

If *value* is negative:

    Let E = edges of X expanded toward the inside by *value*

    L = L NOT E

Else

    Let E = edges of X expanded toward the outside by *value*

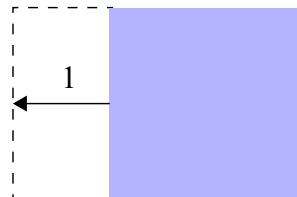
    L = L OR E

An example of this behavior is described in [Figure 4-117](#).

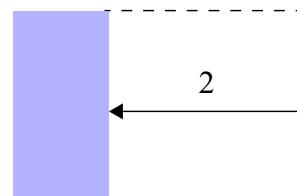
**Figure 4-117. Grow SEQUENTIAL Example**

Edge expansion [ ] Input layer   
(layer1)

GROW layer1 LEFT BY 1 RIGHT BY -2 TOP BY 1 BOTTOM BY -2 SEQUENTIAL



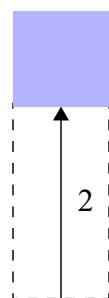
[1] Left edges of the input layer are expanded to the outside by 1 and added to the input layer.



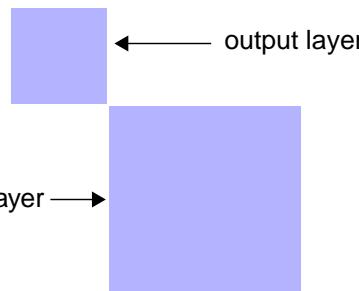
[2] Right edges from the output of [1] are expanded to the inside by 2 and subtracted from the output of [1].



[3] Top edges from the output of [2] are expanded to the outside by 1 and added to the output of [2].



[4] Bottom edges from the output of [3] are expanded to the inside by 2 and subtracted from the output of [3].



## Grow

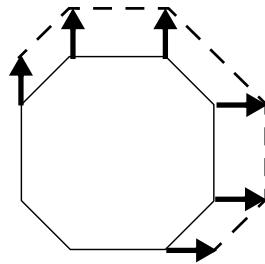
---

The SEQUENTIAL keyword is useful for replacing combinations of Grow and Shrink operations. The following two checks are equivalent:

```
GROW_METAL3i_SEQ {  
    GROW METAL3i LEFT BY -2 RIGHT BY 2 TOP BY -4 BOTTOM BY 4 SEQUENTIAL  
}  
  
GROW_METAL3i_SHRINK_GROW {  
    GCOL = SHRINK METAL3i LEFT BY 2 RIGHT BY 0 TOP BY 0 BOTTOM BY 0  
    GCOLR = GROW GCOL LEFT BY 0 RIGHT BY 2 TOP BY 0 BOTTOM BY 0  
    GCOLRT = SHRINK GCOLR LEFT BY 0 RIGHT BY 0 TOP BY 4 BOTTOM BY 0  
    GCOLRTB = GROW GCOLRT LEFT BY 0 RIGHT BY 0 TOP BY 0 BOTTOM BY 4  
}
```

See also [Shrink](#), [Size](#), and [Expand Edge](#).

## Examples



```
grow_A = GROW layerA RIGHT BY 1 TOP BY 1  
// orthogonal edges classified as top or right are moved  
// 1 unit in either the y- or x-direction, respectively.  
// top-and-left edges are moved 1 unit in the y-direction.  
// bottom-and-right edges are moved 1 unit in the x-direction.  
// top-and-right edges are moved 1 unit in the x- and  
// y-directions.edges that match both of the orientation  
// keyword sets undergo expansion.
```

# Hcell

Specification statement

## **HCELL** *layout\_name source\_name*

Used only in hierarchical Calibre nmDRC and nmLVS.

### Parameters

- ***layout\_name***

A required cell name in the layout that corresponds to a cell name in the source. A ***layout\_name*** can contain one or more asterisk (\*) characters; the (\*) character is a wildcard that matches zero or more characters. Names that use wildcards should be enclosed in quotation marks because the \* is a reserved symbol.

- ***source\_name***

A required cell name in the source that corresponds to a cell name in the layout. The asterisk character does not serve as a wildcard for this parameter.

### Description

Specifies the names of a pair of cells that correspond in layout and source. Such cells are otherwise known as hcells. This statement may appear any number of times and information is accumulated.

The effect of this statement in Calibre nmLVS-H is the same as specifying a pair of cell names in an hcell list with the -hcell command line option. For more information, see the -hcell command line option under “[Calibre nmLVS/nmLVS-H](#)” and the “[Hcells](#)” section in the *Calibre Verification User’s Manual*.

---

#### Note



The Calibre xRC tool ignores this statement—you must use an xcell list instead. Refer to the [Calibre xRC User’s Manual](#).

---

In the nmLVS-H comparison module (calibre -lvs -hier), hcells established with the -hcell and -automatch command line options are combined with hcells from Hcell specification statements. All specified hcells by any of these methods are processed as a single concatenated hcell list. The corresponding cells in the layout and source netlists can then be compared hierarchically. For detailed information on generating hcell lists for circuit comparison, see “[Hcell and Hierarchical Analysis](#)” in the *Calibre Query Server Manual*, or “[Performing Hcell Analysis](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In the nmLVS-H connectivity extraction module (calibre -spice), specifying hcells ensures that all hcells are preserved as subcircuits in the extracted layout netlist and are available for use as hcells in the comparison phase. Note that hcell specification may slow down circuit extraction in cases where it would be beneficial to expand those cells (for example due to dense overlap removal). If you want to control hcells separately during connectivity extraction, use [Layout Preserve Cell List](#). You can control how the tool expands hcells during connectivity extraction through the [LVS Auto Expand Hcells](#) statement.

The **layout\_name** and **source\_name** may be the same or different. You can specify a 1-to-n relation by placing a layout name in several Hcell statements with different source names. Similarly, you can specify a n-to-1 relation by placing a source name in several Hcell statements with different layout names. However, n-to-m relations are not allowed. Note that the latter restriction is enforced by the nmLVS hierarchical circuit comparison module (calibre -lvs -hier) but is not enforced by the rule file compiler, the nmLVS-H circuit extraction module, or by Calibre nmDRC-H.

Case sensitivity of hcell names in nmLVS-H is controlled by the [LVS Compare Case](#) specification statement. Specifically, hcell names are case-insensitive by default; hcell names are case-sensitive if you specify LVS Compare Case YES or LVS Compare Case TYPES. This applies in nmLVS-H to circuit extraction as well as to circuit comparison.

While the statement is checked for syntactical correctness at compilation time, cell names are not matched to the input designs during rule file compilation. Failure to match specified cell names results in runtime warnings. The **source\_name** parameter is not read if the run does not require a source netlist.

You can specify hcells in Calibre Interactive—nmLVS. See “[Supplying Hcell Data](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

### Using Wildcards to Specify Hcells

All **layout\_name** cells whose names match a specified wildcard pattern are treated as hcells. In hierarchical Calibre nmDRC, DFM, and nmLVS-H connectivity extraction, hcells are preserved (not expanded automatically) and are exempt from certain layout transformations. In nmLVS-H, hcells are also used as correspondence cells for circuit comparison.

Consider the following Hcell specification using a wildcard:

```
HCELL "A*" A
```

All layout cells whose names begin with A are expected to correspond to the source cell A.

Calibre nmLVS does not allow many-to-many hcell correspondence. This restriction remains in place when wildcards are used for hcell names. However, when wildcards are present, it is not possible to detect from the rule file alone whether many-to-many correspondence exists since this is design dependent. For example, consider the following Hcell specifications:

```
HCELL "A*" A
HCELL "*B" B
```

If there are no layout cells whose names start with A and end on B, then this is a valid combination of Hcell statements. However, if the layout contains cells named “A1B” and “A2B”, then these cells would correspond to both source cells A and B, thus creating many-to-many correspondence. Therefore, when wildcards are used in Hcell specifications, it is generally impossible to determine whether a given hcell list, by itself, is valid or not. Special care must be taken when creating Hcell specifications, since a specification can pass testing and be released, but still lead to an hcell error on a particular design later.

## Calibre nmDRC-H Use of Hcell Statements

**Note**

For Calibre nmDRC-H, the Hcell statement is only to allow for the flows described below. Otherwise, Hcell statements should not appear in your nmDRC rule file. Such statements seriously degrade nmDRC-H performance in certain designs.

Use of the Hcell specification statement in a nmDRC rule file is applicable to specialized applications (for example, sizing and Litho operations used for manufacturability) where a nmLVS-H run on the output nmDRC-H mask database is planned. Calibre nmDRC-H observes the Hcell specification statements in the rule file and does not allow expansion of hcells on the layout side. See “[nmDRC-H Use of Hcells](#)” in the *Calibre Verification User’s Manual*.

The [Layout Preserve Cell List](#) statement has similar behavior as the Hcell statement in nmDRC.

See also [LVS Auto Expand Hcells](#), [LVS Exclude Hcell](#), [LVS Cell List](#), and [LVS Report Warnings Hcell Only](#).

### Examples

```
//      layout      source

// one-to-one correspondence
HCELL alu      alu
HCELL buf      buffer

// a one-to-two correspondence
HCELL cellA    cellB
HCELL cellA    cellC

// a two-to-one correspondence
HCELL cellD    cellF
HCELL cellE    cellF

// a many-to-one correspondence with wildcard
HCELL "cellG*" cellJ
```

## Holes

Layer operation

**HOLES** *layer* [*constraint*] [INNER] [EMPTY]

### Parameters

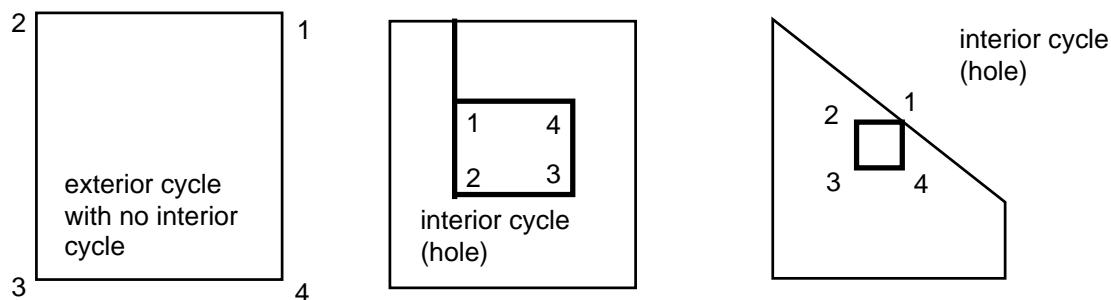
- *layer*  
An original layer or layer set, or a derived polygon layer.
- *constraint*  
One of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. This constraint limits hole selection to those whose area in user units squared satisfies the constraint.
- INNER  
Optional keyword that prevents holes containing other holes from being output.
- EMPTY  
Optional keyword that prevents output of holes that are [Not Outside](#) a *layer* polygon.

### Description

Forms a layer consisting of all polygons that fit exactly inside of *layer* polygon holes.

The tool divides a merged polygon into one or more non-intersecting cycles of vertices. The cycle containing the right-most vertex is called the exterior cycle. All other cycles are called interior cycles. (See [Donut](#) for a complete description of cycles.) A polygon with *only an exterior cycle* has no holes. Otherwise, the polygon has interior cycles and, therefore, holes. Figure 4-118 shows the two kinds of cycles.

**Figure 4-118. Holes Operation**



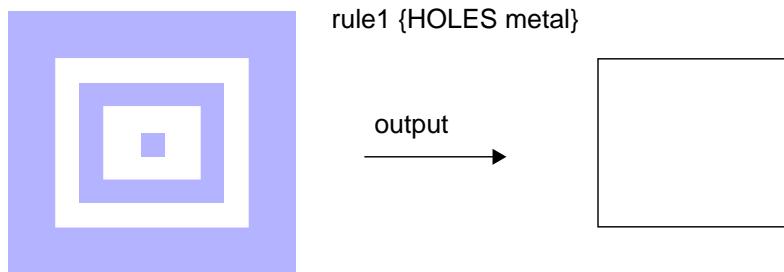
If you specify a *constraint*, it acts as an area filter. Holes are output if their areas in square user units meet the *constraint*.

Here are the possible behaviors for this operation:

- **HOLEs layer [constraint]:**
  - a. Find unmerged holes.
  - b. Test the area of the unmerged holes against the *constraint*, if specified, and filter those whose area does not satisfy the *constraint*.
  - c. Merge results from step b.

For example, notice that holes within holes are merged in [Figure 4-119](#). This is the default behavior:

**Figure 4-119. Holes metal**



**Holes layer** should be used with care as it produces merged output. Specifically, holes within holes are merged away. This can lead to unexpected results. Other keywords allow selection of nested holes.

If a *constraint* is used, then holes are selected by area prior to merging the output. For example, if a hole has area 10 and it lies within another hole of area 100, then a constraint of “< 20” would output the smaller hole. The unconstrained Holes operation would output the larger hole because the smaller hole would be merged away.

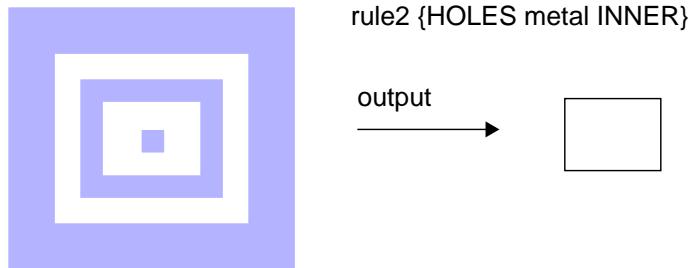
- **HOLEs layer INNER [constraint]:**
  - a. Find unmerged holes.
  - b. Filter out unmerged holes that contain any other unmerged holes. The results from this step are merged by definition.
  - c. Test the area of the results from step b against the *constraint*, if specified, and filter those whose area does not satisfy the *constraint*.

## Holes

---

For example, notice in [Figure 4-120](#) only the innermost hole is merged and output. INNER finds the innermost hole that lies within other holes:

**Figure 4-120. Holes metal INNER**

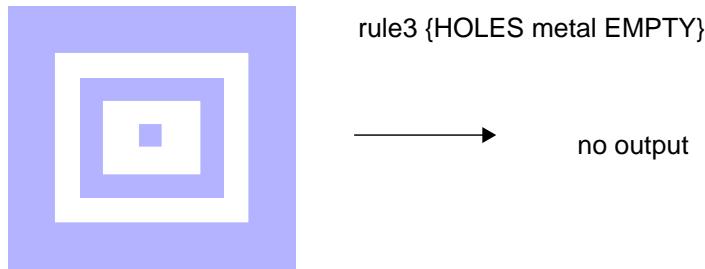


The INNER keyword allows selection of holes not containing other holes before any layer merging occurs. This is useful in isolating metal slots, for example.

- **HOLES *layer* EMPTY [constraint]:**
  - a. Find unmerged holes.
  - b. Merge them.
  - c. Filter output from step b which is [Not Outside](#) the input *layer*.
  - d. Test the area of the results from step c against the *constraint*, if specified, and filter out those whose area does not satisfy the *constraint*.

For example, initially the EMPTY keyword creates a merged hole like [Figure 4-119](#). However, this result is Not Outside the metal layer, so the initial result is filtered and there is no output. EMPTY outputs holes that have no polygons from the input *layer* lying inside the holes:

**Figure 4-121. Holes metal EMPTY**



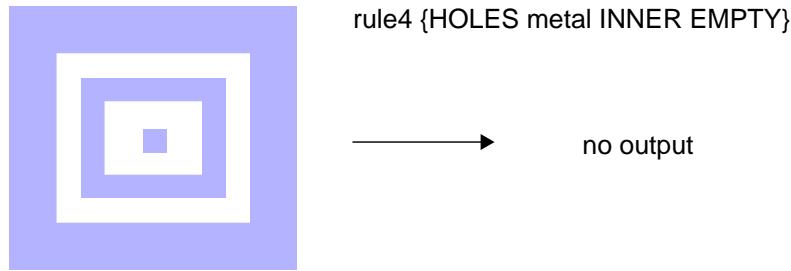
The EMPTY keyword is rarely useful when INNER is not also specified.

- **HOLES *layer* INNER EMPTY [constraint]:**
  - a. Find unmerged holes.
  - b. Filter unmerged holes which contain any other unmerged holes. The results from this step are merged by definition.

- c. Filter output from step b that is Not Outside the input *layer*.
- d. Test the area of the results from step c against the *constraint*, if specified, and filter those whose area does not satisfy the *constraint*.

For example, initially this keyword set creates a merged hole like [Figure 4-120](#). However, this result is Not Outside the metal layer, so the initial result is filtered out and there is no output:

**Figure 4-122. Holes metal INNER EMPTY**



Use of INNER and EMPTY together is rare. It may be of some value in certain metal slot checks.

## Examples

### Example 1

```
// Select all contacts that are exactly in a hole of diffusion:  
X = HOLES diff // First find diffusion holes  
Y = X INSIDE contact  
// Next pick contacts in the holes  
contacts_in_holes = contact INSIDE Y
```

### Example 2

If a hole of area 10 is within another hole of area 100, then this operation:

**HOLES layer < 20**

outputs only the smaller hole. The unconstrained Holes operation would output only the larger hole since the smaller hole would be merged away. (The selection by constrained area occurs before merging.) The INNER keyword can be useful in these situations.

### Example 3

The constrained Holes operation is valuable for performing enclosed area checks. For example, if the enclosed area of metal must be greater than 2, then this operation:

**HOLES metal < 2**

is generally more efficient (and more accurate) than this one:

**AREA (( EXTENT ) NOT metal) < 2**

and is correct as long as a hole of larger dimension cannot contain a shape which removes enough of the hole to generate an error. This is generally the case with most processes.

# Include

Specification statement

**INCLUDE** *filename*

## Parameters

- *filename*

A required pathname of a file containing elements to be used as part of an aggregate rule file. The form of the filename can be a directory component followed by a filename or a variable specified in the shell environment, followed by a filename.

For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in chapter 2, “Key Concepts”.

## Description

Includes a file with the specified *filename* into the current rule file. This statement is equivalent to writing the entire text of the included file in the parent rule file. You can specify this statement any number of times within your rule file. There is no limit to the number of nested Include statements; an included rule file can contain an Include statement. The Include statement must be the only entity on a single line in a rule file.

You may use portions of SVRF syntax in an Include file. In other words, the content of an Include file can be any allowed SVRF syntax.

When the tool compiles a rule file, it processes all Include statements first. That is, a rule file is first *flattened* when you compile it. The tool reports line numbers relative to the included rule file when it generates compiler errors. Include statements are processed before /\*...\*/ comments but not before // comments.

The format of *filename* can be either of the following:

```
INCLUDE pathname/filename  
INCLUDE $variable/filename
```

where *pathname* is a hard or relative path to *filename*, *variable* is a pathname defined in the shell environment prior to invocation, and *filename* is the name of the rule file to be included. When you specify a *variable* in an Include statement, it must be preceded by a dollar sign (\$). Note, however, that ICverify applications cannot follow Include pathnames containing environment variables.

You can specify included rule files in Calibre Interactive through the Include tab, which is a part of the GUI Options pane. See the “[Including Other Rule Files and SVRF Commands in Calibre Interactive](#)” in the *Calibre Interactive and Calibre RVE User’s Manual* for details.

You can cause the Calibre tool to echo the main rule file and the text of all included rule files to the run transcript by setting the environment variable CALIBRE\_ECHO\_RULE\_FILE to a non-null value. The actual Include statements and code that is not executed due to preprocessor directives are commented out in the transcript.

The Include statement may not be used to include compile-time TVF rule files in an SVRF rule file. The tvf::INCLUDE statement, which is used in TVF rule files and performs a similar function as the Include specification statement, can only be used to include SVRF rule files in a TVF rule file flow. To include TVF rule files in a TVF control file, use the Tcl “source” command.

## Examples

### Example 1

This statement:

```
INCLUDE "/usr2/scratch3/howard/block/rules/rules.antenna"
```

is equivalent to the following statement:

```
INCLUDE "$dir/rules.antenna"
```

where dir is defined in a shell environment as:

```
/usr2/scratch3/howard/block/rules
```

Note that in the second Include statement dir is preceded by a dollar sign (\$).

### Example 2

This example shows how the Include statement can be used as an embedded reference to a filename:

```
DRC RESULTS DATABASE
INCLUDE pathname
```

Here the Include statement references a file that contains the definition of the DRC Results Database file. The *pathname* file must contain a valid file path.

### Example 3

This example shows how the Include statement can be used as an embedded reference to a filename containing a list of cells. The filename is used in a Variable statement and passed to a layer operation. The list of cells in the file is then read.

Assume you have a file called cells.txt with the following entries (notice the quotation marks):

```
"cell1"
"cell2"
"cell3"
```

You can pass the filename as a variable, like this:

```
VARIABLE my_cells
INCLUDE cells.txt

x = EXTENT CELL my_cells
```

## Inductance MICheck

Layer operation

**INDUCTANCE MICHECK** (*layer* [*layer...* BY *via* [*via...*] ]) **constraint tolerance**  
**PRINT** *filename*

Used only in Calibre xL ERC.

### Parameters

- **()**  
A required pair of parentheses that surround the layers and vias.
- ***layer***  
A required original or derived layer, optionally followed by other layers. The layer names must be included in a **PEX Elayer** statement, which means the layers cannot be derived within the rule.
- **BY *via***  
An optional keyword set specifying at least one via layer. The layers along with any vias are used to obtain the geometries of the intentional inductors for which the mutual inductance coefficients are calculated. Any branches are ignored.
- ***constraint***  
A required symbol, either < or >, specifying the comparison.
- ***tolerance***  
A required value specifying the acceptable absolute value of the magnetic noise parameter K between the intentional inductors.
- **PRINT *filename***  
A required keyword and parameter that specify the name to use for the report file and the prefix for the Inductance ERC files.

### Description

Inductance MICheck provides an estimate of the magnetic noise parameter K between two devices, such as intentional inductors. K is defined as the ratio of the mutual inductance between two intentional devices normalized to the geometric mean of the two self inductances. Because the values of the mutual inductance and, therefore, the values and K depend on the relative positioning of the devices, it is often desirable during design exploration to study different geometrical configurations and analyze the electrical quality of the resulting layout.

Inductance MICheck does not run as an LVS or DRC rule check. Consequently, rule checks containing the statement are ignored if specified as part of **ERC Select Check** or **DRC Select Check**.

## Report File and RVE Database Output

The *filename* report file is an ASCII file that lists the self inductances, mutual inductances, and the noise parameter value K for each pair of intentional inductors extracted during an Inductance ERC run.

The Inductance ERC files contain the geometries of the intentional inductors that violated the Inductance MICheck rule. There is one Inductance ERC file per violation. You can use the Calibre RVE viewer to view these geometries in the Inductance ERC files.

**Note**



If two Inductance MICheck rules have the same PRINT *filename*, the report file will contain data corresponding to the last Inductance MICheck rule executed.

The report file has the following format:

```
Geometry Set 1 :- L0 = value nH, L1 = value nH, K = value,
    MI = value nH Output Rdb-Layer = mi_chk_out_0_1
Geometry Set 2 :- L0 = value nH, L1 = value nH, K = value,
    MI = value nH Output Rdb-Layer = mi_chk_out_0_2
Geometry Set 3 :- L0 = value nH, L1 = value nH, K = value,
    MI = value nH Output Rdb-Layer = mi_chk_out_0_3
```

where

- L0 and L1 are self inductances.
- K is the mutual inductance coefficient.
- MI is the mutual inductance in nanohenrys.
- Output Rdb-Layer is the name of the layer that contains the geometries of the intentional inductors. This is also the name of the generated Inductance ERC file.

## Limitations

The Inductance MICheck rule has the following limitations:

- All structures for which inductance is calculated must be linearly connected; that is, the structures may bend, but they cannot branch.
- Signal frequency affects inductance. The Inductance MICheck rule is most accurate for frequencies up to a few GHz.
- The Inductance MICheck rule can only be used as an output layer operation.
- You must look at the report file and the output Rdb-Layer referenced in the report file to find self and mutual inductance values.

## Error Messages

The following error message displays when the layer names used in the Inductance MICheck rule are not specified in the rule file. Edit your rule file to include the layers names in a PEX Alias or PEX Elayer statement, and then rerun Inductance ERC.

```
ERROR: Error PEX35 on linenumber of rulefile - INDUCTANCE MICHECK may only  
use derived layers with profile information.
```

## Examples

### Example 1: Single-Level Intentional Inductor Interaction

In Example 1, the Inductance MICheck rule is used in its simplest form. All geometries to check are on a single layer, Inductors.

```
{ Inductance MICHECK ( Inductors ) > 0.3 print mi_chk_out1 }
```

The working directory will contain two or more files if there were more pairs.

- *mi\_chk\_out1*, which you can view using a text editor.
- *mi\_chk\_out1\_0\_1*, which you can view using the Calibre RVE viewer.

### Example 2: Intentional Inductor-to-Inductor Interaction

In Example 2, the Inductance MICheck rule checks for intentional inductors with unacceptable noise parameters. Assuming that m4I and m5I are derived layers that are connected by the via layer named via45, the m4I\_m5I\_spirals layer contains intentional inductor pairs with a mutual inductance coefficient greater than 0.1.

```
{ Inductance MICHECK ( m4I m5I BY via45 ) > 0.1 print m4I_m5I }
```

If the absolute value of K of the intentional inductor pairs is larger than 0.1, the Inductance MICheck rule is violated and the m4I\_m5I\_spirals layer contains the geometries that violated the rule.

If there are two inductors in the layers defined by ( m4I m5I BY via45 ), the *m4I\_m5I* report file in your working directory would look similar to the following:

```
Geometry Set 1 :- L0 = 0.64 nH, L1 = 1.29 nH, K = 0.103, MI = 0.095 nH  
Output Rdb-Layer = m4I_m5I_0_1
```

The working directory also contains the *m4I\_m5I\_0\_1* file, which you view through the Calibre RVE viewer.

# Inductance Wire

Specification statement

**INDUCTANCE WIRE** *layer* { '['*equation*' ] | '['*n1 n2*' ] }

Used only in Calibre xL.

## Parameters

- ***layer***  
A required argument specifying the bonding wire layer.
- '['*equation*' ]'  
A required arbitrary equation specifying how to calculate the self inductance of bonding wires, which is a function of the length() parameter (in microns) of the bonding wire. For information on the syntax conventions for the equation, refer to the syntax conventions for capacitance and resistance equations described in the Built-in Language chapter in the Standard Verification Rule Format (SVRF) manual. The equation must be enclosed in brackets.
- '['*n1 n2*' ]'  
A required set of two values specifying the parameters to use in the equation  $L = n1 * \text{length}() + n2$  for calculating bonding wires, where length is the length (in microns) of the bonding wire. The parameters must be enclosed in brackets.

## Description

This optional statement specifies the parameters to use for calculating the self inductance of bonding wires. Values are specified in inductance units, which defaults to picohenries and is specified using the [Unit Inductance](#) statement. If the Inductance Wire statement is not included in the SVRF rule file, the inductance of bonding wires is calculated using partial inductance.

## Examples

The first statement instructs Calibre xL to calculate the self inductance of bonding wires on the BW1 layer using the equation  $L = n1 * \text{length}() + n2$ , where *n1* is 2.3 and *n2* is 3.5. The second statement instructs the Calibre xL tool to calculate the self inductance of bonding wires on the BW2 layer using the arbitrary equation specified.

```
INDUCTANCE WIRE BW1 [ 2.3 3.5 ]
INDUCTANCE WIRE BW2 [
    PROPERTY L
    if (length() > 2.0) {
        L = 5.0 * length()*(log(0.18*length()) + 0.75) + 10.0
    }
    if (length() <= 2.0 ) {
        L = 4.2 * length()
    }
]
```

## Inside

Layer operation

**INside *layer1* *layer2***

### Parameters

- ***layer1***  
An original layer or layer set, or a derived polygon layer.
- ***layer2***  
An original layer or layer set, or a derived polygon layer.

### Description

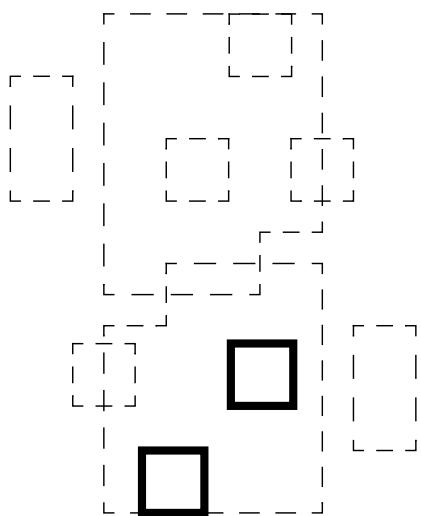
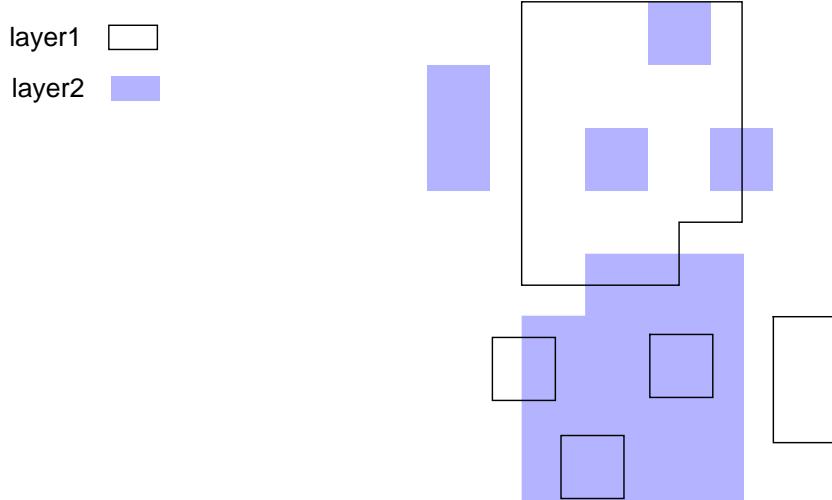
Selects all ***layer1*** polygons that share their entire areas with ***layer2*** polygons. If a ***layer1*** polygon shares all of its area with a ***layer2*** polygon and the two polygons have coincident edges, the Inside condition is satisfied.

If either input layer is empty, there is no output.

See also [Not Inside](#), [Enclose](#), [Inside Cell](#), [Inside Edge](#), [Interact](#), [Outside](#), and [Not Outside](#).

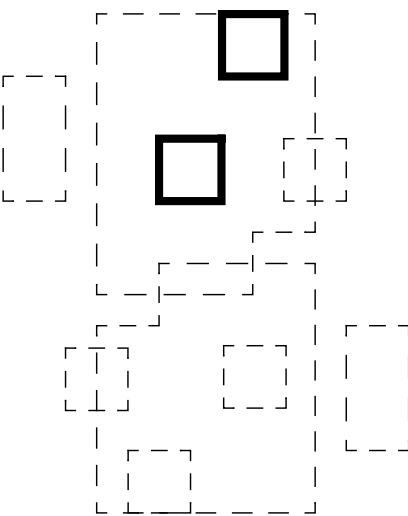
### Examples

Figure 4-123 shows two Inside operations. Operation 1 selects all *layer1* polygons that lie completely inside any *layer2* polygons (this includes coincident edges). Operation 2 reverses the layer order; therefore, it selects all *layer2* polygons that lie completely inside *layer1* polygons (again, this includes coincident edges).

**Figure 4-123. Inside**

inside layer1 layer2  
//also layer1 inside layer2

Operation 1



inside layer2 layer1

Operation 2

## Inside Cell

Layer operation

**INSIDE CELL** *layer* *cell\_name* [*cell\_name* ...] [PRIMARY ONLY]  
[WITH MATCH [GOLDEN] [RULE *rule\_name*] ...]  
[WITH LAYER *layer2*]

Used primarily in hierarchical Calibre applications.

### Summary

Selects all polygons from the input *layer* that are inside the specified cells, and the subhierarchies of the specified cells. You must use the parameter order as shown to avoid ambiguity.

### Parameters

- *layer*

A required original layer.

- *cell\_name*

A required name of a cell. You can specify *cell\_name* any number of times in one statement. Cell names are case-sensitive. The *cell\_name* can be a string variable (see [Variable](#)).

The *cell\_name* parameters can contain one or more asterisk (\*) wildcard characters, where the \* character matches zero or more characters. When using \*, enclose the cell name in quotes; otherwise, a compilation error occurs because the asterisk is a reserved symbol.

The string CALIBRE\_TOP\_CELL (case-insensitive) is understood to be the top-level cell in the database.

- PRIMARY ONLY

An optional keyword that instructs the tool to output geometry only from the top level of the specified cells, and to exclude shapes that exist in the subhierarchy of the specified *cell\_name* parameters. The proper sub-hierarchy of a cell is ignored, unless it coincides with the top level of another cell in the *cell\_name* parameter list.

- WITH MATCH [GOLDEN] [RULE *rule\_name*] ...

An optional keyword set that allows a placed cell to be treated as a *cell\_name* parameter to the operation if this cell geometrically matches another cell (unplaced) that is already specified as a *cell\_name* parameter. The WITH MATCH keyword is particularly useful for matching cells to so-called “golden” cells, when the exact name correspondence may not be known. This keyword set may only be used in hierarchical Calibre applications. The following additional keywords may be specified:

- GOLDEN

Causes the operation to attempt to match only cells that appear in designs specified with [Layout Path](#) or [Layout Path2](#) statements that also use the GOLDEN keyword. The GOLDEN keyword may only be specified with the WITH MATCH keyword.

- RULE *rule\_name*

Causes the operation to use [Layout Cell Match Rule](#) statements having a matching *rule\_name* in order to define the cells that are geometrically equal. The Layout Cell Match Rule algorithm is used instead of the default algorithm for determining geometrically equal cells. More than one RULE keyword set may be specified, and this keyword set may only be specified with WITH MATCH. Each *rule\_name* that matches a Layout Cell Match Rule statement is used. Duplications of *rule\_name* are ignored; failure to match a corresponding *rule\_name* causes a compiler error.

- WITH LAYER *layer2*

An optional keyword that limits selection to the specified cells that have any geometry on *layer2* (which must be an original layer) in their immediate hierarchy.

## Description

Selects all polygons from the input *layer* that are inside the specified cells, and the subhierarchies of the specified cells. You must use the parameter order as shown to avoid ambiguity.

The tool selects only the shapes that are instantiated inside of a cell. A shape that is inside the boundary of every placement of a cell, but at a higher level of hierarchy, is not selected.

This statement is used primarily for hierarchical Calibre applications. It can be used in flat Calibre, but its usefulness is limited.

By default, a warning is issued if a *cell\_name* parameter cannot be found in the layout, but the run proceeds. You can control the severity of this case through the [Layout Input Exception Severity INSIDE\\_CELL](#) setting.

The PRIMARY ONLY keyword selects only top-level geometry from specified cells. Output layers generated using this option in a hierarchical run *are not merged*. Multiple abutting polygons may appear in the output in this case.

The WITH MATCH keyword allows placed cells to be included as part of the Inside Cell calculation based upon geometric matching of another cell used as a *cell\_name* parameter. This feature should only be used with validated cells serving as *cell\_name* parameters in a hierarchical run. Note that specifying “non-golden” cells (cells that have not been previously validated) can result in warnings or errors during the Calibre run.

Before discussing how WITH MATCH works, the term *geometrically equal* is defined. Two cells A and B are said to be *geometrically equal* under the following conditions:

1. Flatten and merge *layer* in A and its sub-hierarchy into the coordinate space of A.
2. Flatten and merge *layer* in B and its sub-hierarchy into the coordinate space of B.

3. Perform a Boolean XOR of the results of Steps 1 and 2. If there are no results of the XOR, then A and B are geometrically equal.

This definition does not include text objects.

The output of WITH MATCH is as follows.

**INSIDE CELL layer A<sub>1</sub> A<sub>2</sub> ... A<sub>m</sub> WITH MATCH ...**

where m >= 1, is made equivalent to

**INSIDE CELL layer B<sub>1</sub> B<sub>2</sub> ... B<sub>n</sub>**

where n >= 0. Each B<sub>j</sub> is the name of a placed cell (the name may or may not be known) that is geometrically equal to some unplaced cell A whose name matches some A<sub>i</sub> with (possibly) wildcards. The geometry on *layer* that appears in all such cells B<sub>j</sub> is output.

The RULE *rule\_name* keyword causes the matching [Layout Cell Match Rule rule\\_name](#) statement to be used to determine which cells are geometrically equal.

If the GOLDEN keyword is specified, then the unplaced cells A<sub>i</sub> that the operation attempts to match come from designs specified in Layout Path[2] GOLDEN statements.

Consider this example:

Suppose you have a rule for which the poly spacing is different for the memory and non-memory portions of your design. You want to apply two spacing checks, depending on whether the cell is a memory cell or not. In the file golden.gds, the memory cells appear in the cell bitcel.golden. In layout.gds, the names of the memory cells are not known, so you want to match the memory cells geometrically to bitcell.golden.

```
LAYOUT PATH layout.gds golden.gds //concatenate the layouts.  
// layout cell names are not expected to match.  
LAYOUT ALLOW DUPLICATE CELL NO // default  
  
// Find the poly in memory cells in layout.gds by geometric matching  
// with golden.gds. Specifying bitcel.golden isn't expected to match  
// a cell name in layout.gds; however, it is the known memory cell name  
// in golden.gds.  
  
poly_in_bitcel = poly INSIDE CELL bitcel.golden WITH MATCH  
  
40X { @ poly spacing is 0.3, except in bit cells where it is 0.2.  
  
    X1 = EXT poly < 0.3 REGION EXTENTS  
    // apply 0.3 spacing measurement for the entire design  
  
    X2 = EXT poly_in_bitcel < 0.3 REGION EXTENTS  
    // apply 0.3 spacing in memory cells;  
    // note X1 has already calculated it so there is no runtime penalty.  
  
    X3 = X1 NOT X2  
    // spacing errors that are not in memory cells  
  
    X3 OR ( EXT poly_in_bitcel < 0.2 REGION EXTENTS )  
    // apply a different measurement for memory cells and OR with  
    // other cells' results
```

```
}
```

In ICVerify applications, this operation produces an empty layer with a warning.

See also [Not Inside Cell](#).

## Examples

### Example 1

The following statement selects all metal from cells ramcell and romcell, including the subhierarchies of these cells.

```
x = INSIDE CELL metal ramcell romcell
```

The corresponding operation excludes metal from selection which exists in the subhierarchies of cells ramcell and romcell. If romcell is instantiated in the subhierarchy of ramcell, then metal at the primary level in cell romcell is still selected by the operation.

```
x = INSIDE CELL metal ramcell romcell PRIMARY ONLY
```

### Example 2

The following statement selects all polygons from metal1 that are inside any cell, including the subhierarchies of any cell. The WITH LAYER sram parameter limits selection to those cells that have any geometry on layer sram in their immediate hierarchy.

```
metal1_sram = metal1 INSIDE CELL '*' WITH LAYER sram
```

### Example 3

This example is based upon the one shown in the Description section. This shows how the GOLDEN option might be used when golden cells are not to be concatenated with the primary design:

```
LAYOUT PATH layout.gds
LAYOUT PATH golden.gds GOLDEN // do not concatenate bitcel.golden with
// layout.gds

// Find the poly in memory cells in layout.gds by geometric matching
// with golden.gds.

poly_in_bitcel = poly INSIDE CELL bitcel.golden WITH MATCH GOLDEN
```

## Inside Edge

Layer operation

**INside EDGE *layer1 layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
An original layer or layer set, or a derived polygon layer.

### Description

Selects all *layer1* edges or edge segments that lie completely inside *layer2* polygons. Any *layer1* edges that are coincident with *layer2* edges are not output.

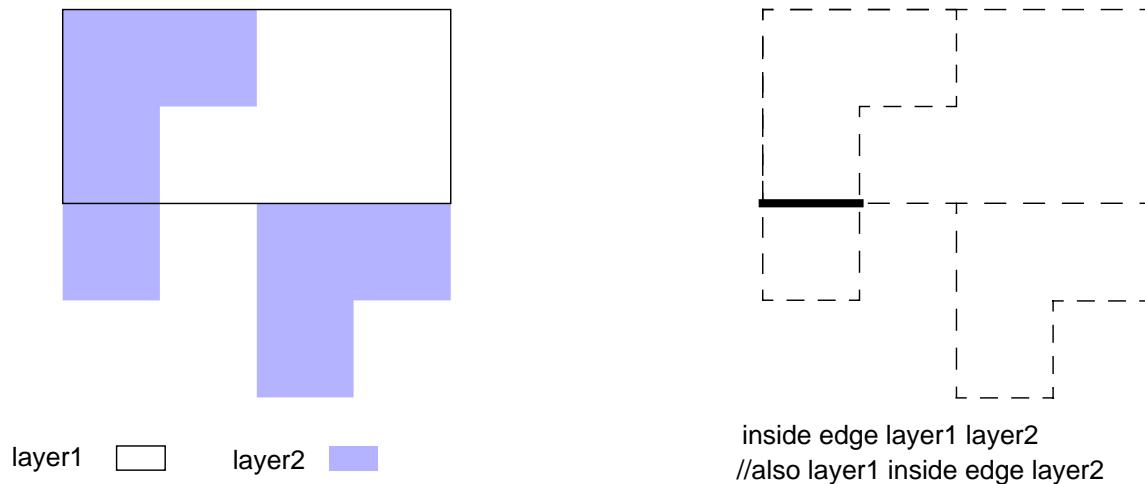
See also [Coincident Inside Edge](#), [Not Inside Edge](#), [Outside Edge](#), [Not Outside Edge](#), [Touch Inside Edge](#), and [Inside](#).

### Examples

#### Example 1

Figure 4-124 shows an Inside Edge operation that selects all *layer1* edges or edge segments that lie completely inside *layer2* polygons, even if the endpoint(s) of the *layer1* edges or edge segments touch the inside of the *layer2* edge(s).

**Figure 4-124. Inside Edge**



**Example 2**

Layer derivation.

```
// Select gate edges along poly. Note the gate layer is not
// explicitly generated.
poly_gate_edges = poly INSIDE EDGE diff
```

## Interact

Layer operation

**INTERACT** *layer1* *layer2* [*constraint* [BY NET] [EVEN | ODD]]  
[SINGULAR {ALSO | ONLY}]

### Summary

Selects polygons based upon the type and number of intersections they have with polygons on another layer.

### Parameters

- ***layer1***  
An original layer or layer set, or a derived polygon layer.
- ***layer2***  
An original layer or layer set, or a derived polygon layer.
- ***constraint***  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. You specify the constraint to limit the selection of *layer1* polygons according to the number of *layer2* polygons with which the interaction occurs. The constraint should contain positive integers.  
The constraint == 0 does not produce any output. Use [Not Interact](#) instead.
- **BY NET**  
An optional keyword, used with the *constraint* parameter, which specifies the selection of a *layer1* polygon based upon the number of distinct nets it interacts with on *layer2*. The connectivity of *layer2* is required and is checked at compilation time.
- **EVEN**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an even number. May not be used with ODD.
- **ODD**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an odd number. May not be used with EVEN.
- **SINGULAR {ALSO | ONLY}**  
Optional keyword set that indicates how to handle point-to-point and point-to-edge interactions. If neither of these keyword sets are used, point-to-point and point-to-edge interactions are ignored. May not be specified with BY NET.

SINGULAR ALSO — Selects all polygons from *layer1* that intersect polygons on *layer2* in *any* manner, including outside point-to-point and point-to-edge interaction. If a

*constraint* is specified, then *layer1* must intersect the specified number of *layer2* polygons in any manner.

**SINGULAR ONLY** — Selects all polygons from *layer1* that intersect polygons on *layer2* by outside point-to-point and point-to-edge interaction, regardless of any other interactions. If a *constraint* is specified, then *layer1* must intersect the specified number of *layer2* polygons in point-to-point and point-to-edge interaction. Note that a *layer1* polygon can overlap or share edge segments with *layer2* polygons and still be output; it is the presence of singularities that must be satisfied for output to occur.

## Description

Selects all *layer1* polygons that either share some or all of their area with a *layer2* polygon, or that are outside all *layer2* polygons and have a coincident outside edge (see [Coincident Outside Edge](#) for a description) or edge segment with a *layer2* polygon. If either input layer is empty, there is no output.

## BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that meet the constraint are output.

## EVEN and ODD Keywords

The EVEN and ODD keywords modify the interpretation of the *constraint*. For example:

**layer1 INTERACT layer2 >3 <9 EVEN**

This operation selects polygons from *layer1* that interact with four, six, or eight polygons from *layer2*.

A constraint must be specified with these keywords. It is not useful to specify these keywords if your *constraint* does not allow for the number of polygons to meet the even or odd criterion, such as with == constraints.

## SINGULAR Keyword

The SINGULAR ALSO and SINGULAR ONLY keywords enable point-to-point and point-to-edge interactions to be considered as meeting the Interact criteria. In the former case, point-to-point and point-to-edge interactions may be present for *layer1* polygon output to occur, but they do not have to be present; in the latter case such interactions must be present for *layer1* polygon output to occur.

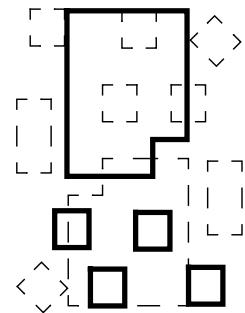
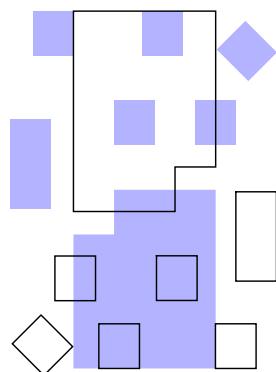
See also [Cut](#), [Enclose](#), [Inside](#), [Net Interact](#), [Not Interact](#), [Not Outside](#), and [Touch](#).

**Examples****Example 1**

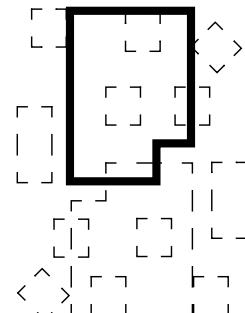
Figure 4-125 shows examples of the Interact statement. Operation 1 selects all polygons on layer1 that interact with a polygon of layer2. Operation 2 selects all polygons on layer1 that interact with more than one polygon from layer2.

**Figure 4-125. Interact**

layer1      
 layer2   



```
interact layer1 layer2
//also layer1 interact layer2
Operation 1
```



```
interact layer1 layer2 > 1
Operation 2
```

**Example 2**

This example shows uses of Interact BY NET:

```

layer met1 4
layer poly 5
layer cont 6
connect met1 poly by cont

// Displays met1 shapes that INTERACT with more than 1 net on poly.
// This flags configurations I, III, and IV.
rule1{ met1 interact poly > 1 by net }

// Displays met1 polygons that INTERACT with exactly 1 net on poly.
// This flags configuration II.
rule2 { met1 interact poly == 1 by net }

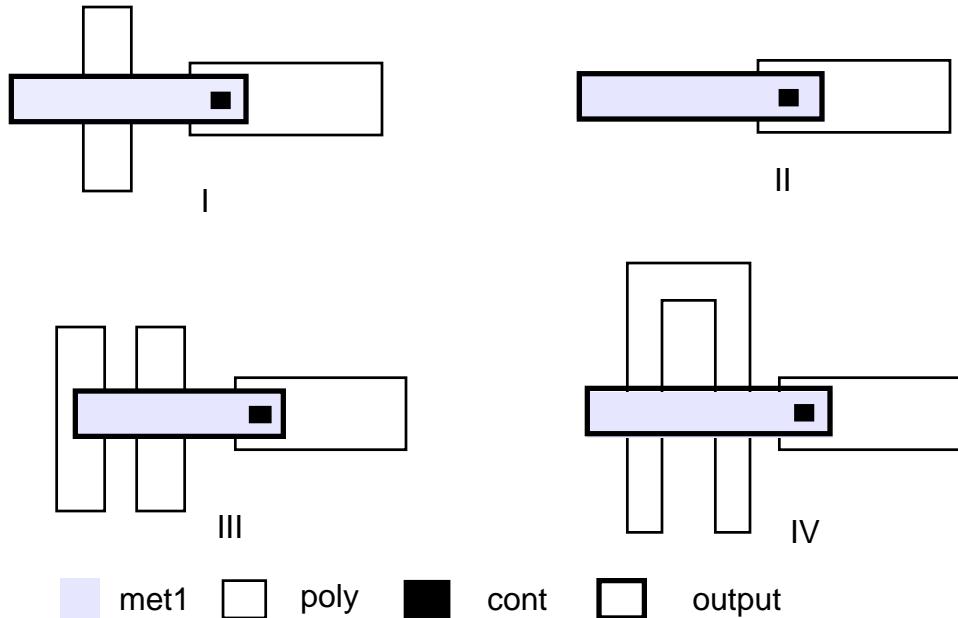
// Displays met1 shapes that INTERACT with exactly 2 nets on poly.
// This flags configurations I and IV.
rule3 { met1 interact poly == 2 by net }

// NOTE: Using BY NET means you are counting the number of
// nets--NOT the number of polygons or touched edges.

```

Figure 4-126 shows example configurations of what these rules find:

**Figure 4-126. Interact BY NET**



## Internal

Layer operation

Single-layer syntax:

```
INTernal layer1 constraint [metric] [orientation_filter] [projection_filter] [angled_filter]
[corner_filter] [intersection_filter] [EXCLUDE FALSE][region_output]
```

Two-layer syntax:

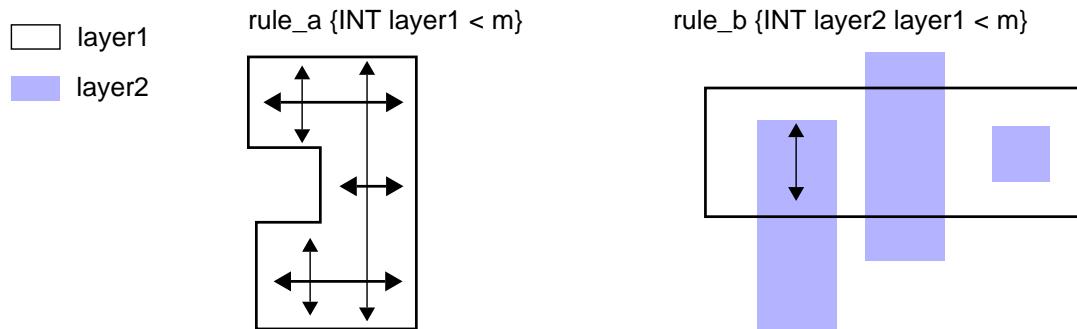
```
INTernal layer1 layer2 constraint [metric] [polygonContainment] [connectivity_filter]
[orientation_filter] [projection_filter] [angled_filter] [corner_filter] [intersection_filter]
[EXCLUDE FALSE] [output]
```

### Summary

For the single-layer syntax, measures the separations between interior-facing sides of edges from the same polygon on *layer1*. For the two-layer syntax, measures the separations between the interior-facing sides of *layer1* edges and the interior-facing sides of *layer2* edges.

The tool outputs measured edge pairs satisfying the given **constraint**. Intersecting edge pairs and single-point interactions are not measured by default. There are multiple secondary keyword sets that control the behavior of the Internal operation for specialized applications. Internal is principally used for width and overlap checks as shown here:

**Figure 4-127. Basic Internal Rule Checks**



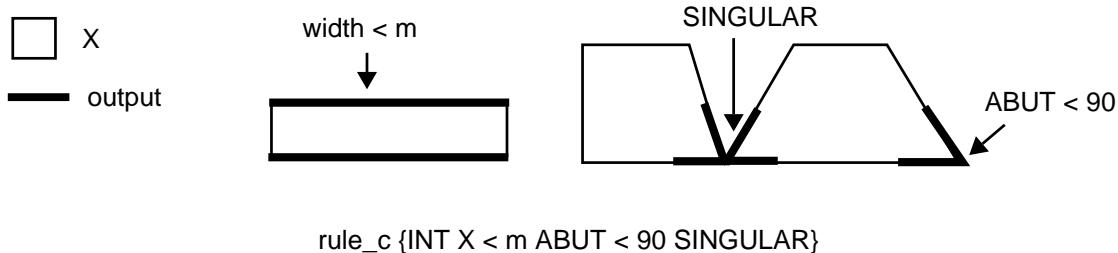
The rule checks shown in Figure 4-127 measure the width between the layer1 edges (single-layer syntax) and overlap of layer 1 and layer2 edges (two-layer syntax). The line segments with arrows indicate where the measurements occur. Notice the order of the layers is unimportant in the two-layer syntax. By default, edge segments from both layers that meet the measurement constraint are output for the two-layer syntax.

The default INTernal operation uses the ACUTE ALSO, PARAllel ALSO, NOT PERPendicular, and NOT OBTUSE keywords with the Euclidean measurement metric (which has no keyword).

Figure 4-128 shows the most common type of Internal rule check. It uses the ABUT and SINGULAR secondary keywords, which are discussed later in this section. ABUT checks

intersecting edges and SINGULAR checks single-point interactions. Note Internal does not check these situations by default.

**Figure 4-128. Typical Internal Check**



If INTernal is used to generate derived polygon layers (as opposed to edge layers), it is best to use the REGION EXTENTS keyword, like this:

```
x = INT lay1 lay2 < 0.04 ABUT < 90 REGION EXTENTS
```

See “[region\\_output](#)” on page 681 for more details about deriving polygon layers with INTernal.

Many fundamental concepts and details of dimensional check operations may be found in the section “Dimensional Check Operations” in the [Calibre Verification User’s Manual](#). You may want to have this reference open while reading about Internal.

Related operations include [Enclosure](#), [External](#), [TDDRC](#), [With Width](#), [DFM Measure](#), and [DRC Tolerance Factor](#).

## Parameters and Description

- **layer1**

An original layer or layer set, or a derived polygon or edge layer. Enclosing *layer1* in brackets [ ]—called positive edge data—indicates that only edges that meet the **constraint** from *layer1* are output. Enclosing *layer1* in parentheses ( )—called negative edge data—indicates that edges from *layer1* that do not meet the **constraint** be output. Positive and negative edge data may be used to form derived edge layers. Refer to “[Edge-Directed Output](#)” in the *Calibre Verification User’s Manual* for additional details.

- **layer2**

An original layer or layer set, or a derived polygon or edge layer. Enclosing *layer2* in brackets [ ]—called positive edge data—indicates that only edges that meet the **constraint** from *layer2* are output. Enclosing *layer2* in parentheses ( )—called negative edge data—indicates that edges from *layer2* that do not meet the **constraint** are output. Positive and negative edge data may be used to form derived edge layers. Refer to “[Edge-Directed Output](#)” in the *Calibre Verification User’s Manual* for additional details.

Positive (or negative) edge data may be used to derive edge layers. When you use the parentheses ( ) parameter in dimensional check operations, you must be careful about making erroneous assumptions. For example, consider the following design rule:

The width of metal must be greater than or equal to 3 microns except where poly and metal overlap by more than 1 micron, then the metal width must be greater than or equal to 4 microns.

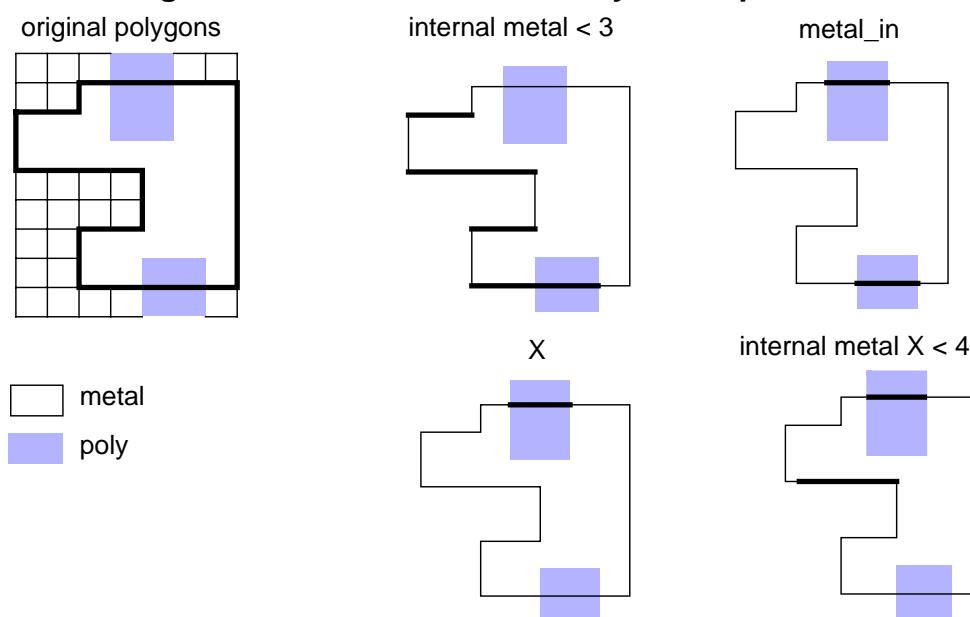
You might be tempted to detect the overlap of poly and metal by more than 1 micron condition by using the following dimensional check operation:

**INTERNAL (metal) poly < = 1**

However, this does not give the expected result because all metal edges that do not conform to the constraint of  $\leq 1$  are selected. This edge data may include metal edges that are not supposed to be candidates for measurement because they are outside of the poly layer. What you need to enforce is the condition that metal and poly must overlap before the wider metal width is checked; that is, metal must be inside of poly. Thus, a better rule check statement to detect the overlap of poly and metal is shown next and illustrated in Figure 4-129.

```
// Metal Width in Poly Overlap Check
METAL004 {
    INTERNAL metal < 3 // normal rule
    metal_in = INSIDE EDGE metal poly
    x = INTERNAL (metal_in) poly <= 1
    INTERNAL metal x < 4
}
```

**Figure 4-129. Metal Width in Poly Overlap Check**



Note that derived error layers, such as the following example (notice the [ ] or ( ) operators are not used, nor is a REGION-type keyword present):

```
x = INT poly diff < 0.1
```

Such derived layers cannot be passed to most other layer operations. However, three DFM operations accept this type of data as input:

- [DFM Analyze](#) — calculates statistics about error data using the COUNT, EC, or EW functions.
- [DFM RDB](#) — writes error layer data to an RDB.
- [DFM Property](#) — filters and/or annotates derived error layers.
- ***constraint***

One of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45, except for  $> a$ ,  $\geq a$ , and  $\neq a$ . Specifies the checking distance, which must have an upper bound. The constraint must contain non-negative real numbers. It is interpreted in user units.

**Note**

 Internal operations written to test a greater than or equal zero ( $\geq 0$ ) condition within a range constraint (as in  $\geq 0 \leq 4$ ), do not return a touching/coincident ( $=0$ ) condition. The ABUT and SINGULAR secondary keywords should be used to check  $=0$  conditions.

Using constraints that are very large in comparison to the typical feature size of a layer, and when the layer is rather dense, can result in poor performance. If your performance for an INTernal operation is poor, check to see if the ***constraint*** values are large in comparison to the expected feature sizes of the input layers. If so, attempt to rewrite your rules to reduce the layer data presented to the INTernal operation. See “[Efficient Width and Spacing Checks](#)” in the *Calibre Verification User’s Manual*.

- *metric*

Secondary keyword set that instructs dimensional check operations to use a specified edge measurement metric when measuring the separation between edges. By default, the measurement metric is Euclidean. You do not need to specify any keyword to get a Euclidean measurement.

For detailed information about all of these metrics, refer to the “Metrics” section of the *Calibre Verification User’s Manual*.

There are three basic metrics in addition to the default.

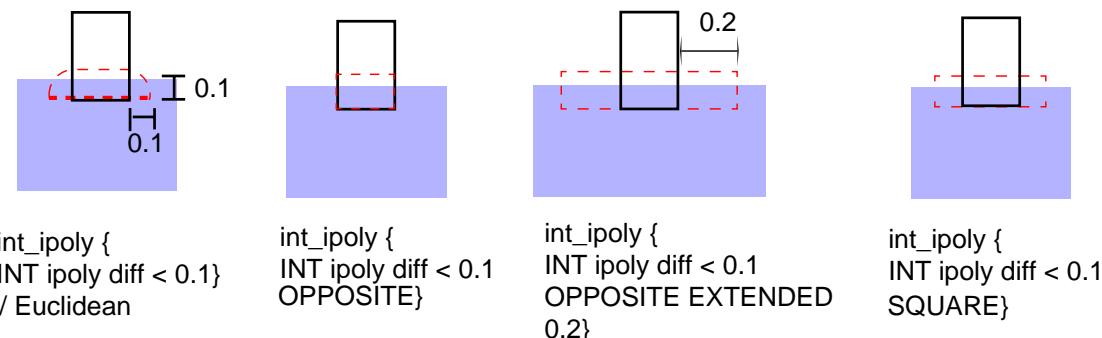
**OPPOSITE** — Specifies extension of the measurement region outward from the edge, but not along the edge, is equal to your **constraint**. It converts the “ $<$ ” constraint to “ $\leq$ ” when measuring intersecting edges. This causes output when intersecting edges abut at 90-degree angles. See the description of the *intersection\_filter* parameter for more discussion of edge intersection.

**OPPOSITE EXTENDED value** — Specifies to use the opposite metric with an extension of the measurement region along the edge. The *value* is an extension distance in user units and is measured along the edge direction; it must be a positive number.

**SQUARE** — Specifies the extension of the measurement region outward along the edge direction and away from the edge is equal to your constraint.

The examples in Figure 4-130 show the four most commonly used measurement regions in red. They show only the measurement regions from ipoly to diff. There are also measurement regions that would be constructed from diff to ipoly.

**Figure 4-130. Measurement Regions**



The following are highly specialized metrics used primarily for optical process correction applications. Refer to the “Metrics” section of the *Calibre Verification User’s Manual* for complete details.

**OPPOSITE SYMMETRIC** — Specialized metric based upon the OPPOSITE metric. Used for adjusting edge output for non-orthogonal edges.

**OPPOSITE FSYMMETRIC** — Specialized metric based upon the OPPOSITE SYMMETRIC metric. Uses a fill-in algorithm that outputs single edges, where disjoint edges might otherwise be output.

**OPPOSITE EXTENDED SYMMETRIC *value*** — Specialized metric based upon the OPPOSITE EXTENDED metric. Used for adjusting edge output for non-orthogonal edges.

**OPPOSITE EXTENDED FSYMMETRIC *value*** — Specialized metric based upon the OPPOSITE EXTENDED SYMMETRIC metric. Uses a fill-in algorithm that outputs single edges, where disjoint edges might otherwise be output.

**OPPOSITE1** — Unidirectional metric based upon the OPPOSITE metric. Similar to OPPOSITE SYMMETRIC but measures edges in one direction from the first input layer to the second input layer.

**OPPOSITE2** — Unidirectional metric based upon the OPPOSITE metric. Similar to OPPOSITE SYMMETRIC but measures edges in one direction from the second input layer to the first input layer.

**OPPOSITE EXTENDED1 *value*** — Unidirectional metric based upon the OPPOSITE EXTENDED metric. Similar to OPPOSITE EXTENDED SYMMETRIC but measures edges in one direction from the first input layer to the second input layer.

**OPPOSITE EXTENDED2 *value*** — Unidirectional metric based upon the OPPOSITE EXTENDED metric. Similar to OPPOSITE EXTENDED SYMMETRIC but measures edges in one direction from the second input layer to the first input layer.

**SQUARE ORTHOGONAL** — Metric used to simulate mask misalignment in the x- and y-directions.

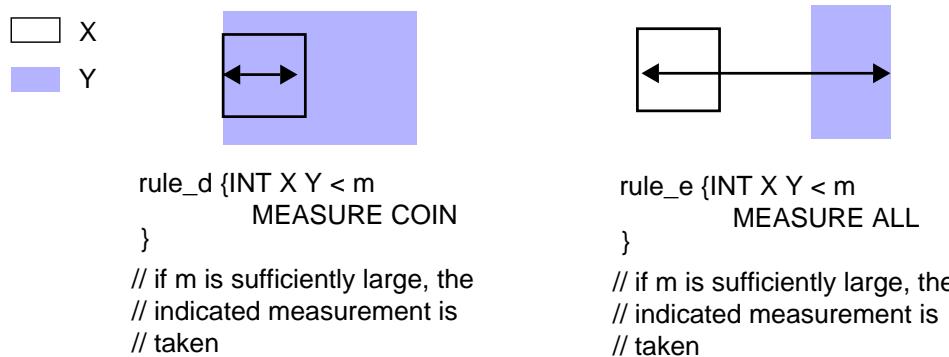
- *polygon\_containment*

The secondary keywords in this set instruct two-layer dimensional check operations to alter the polygon containment criteria when measuring the separation between edges. For more about this topic, refer to the “Polygon Containment Criteria” section of the [Calibre Verification User’s Manual](#).

**MEASURE COINCIDENT** — Relaxes the polygon containment criteria to measure coincident edges: edge A of *layer1* can additionally be outside coincident with edge B from *layer2*.

**MEASURE ALL** — Specifies to ignore the polygon containment criteria. This allows Internal to *see through* polygons that ordinarily it would not. Figure 4-131 shows an example.

**Figure 4-131. MEASURE ALL**



- *connectivity\_filter*

Secondary keyword set that instructs two-layer Internal operations to measure the separation between edges based upon their connectivity. Dimensional check operations ignore connectivity if you do not specify the secondary keywords in this set. Possible choices are:

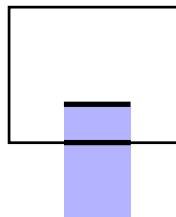
CONNECTED — Specifies to measure only edges from polygons that belong to the same net.

NOT CONNECTED — Specifies to measure only edges from polygons that do *not* belong to the same net.

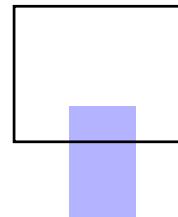
To use these filters, the input *layers* must possess valid connectivity. In [Figure 4-132](#), output occurs based upon connectivity information.

**Figure 4-132. Connectivity-Based Checks**

□ X  
■ Y



```
connect X Y
rule_f {
    int X Y < m
    connected
}
```



```
connect X Y
rule_g {
    int X Y < m
    not connected
}
// produces no output
// due to connect XY
```

For related information, see “Node-Preserving Operations” in the [Calibre Verification User’s Manual](#).

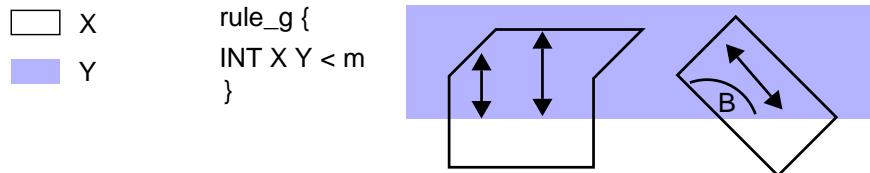
- *orientation\_filter*

Secondary keyword set that instructs the dimensional check operations to measure the separation between edges based upon their appropriate angle or edge orientation. (See the “Appropriateness Criteria” section of the *Calibre Verification User’s Manual*.) You can specify one choice from each of the following four groups in each statement.

The default parameters in this keyword set (PARAllel ALSO, ACUTE ALSO, NOT PERPendicular, NOT OBTUSE) instruct dimensional check operations to measure the separation between the corresponding sides of edges that face each other and that have an appropriate angle of less than 90 degrees.

[Figure 4-133](#) shows the edge orientations that are checked by default using the two-layer Internal operation for an appropriate value m. The single-layer form is similar in its behavior. No other orientations are checked unless you select other filters. Note that intersecting (or abutting) edges as shown by angle B are not checked by default.

**Figure 4-133. Default Edge Orientations Checked by Internal**



The secondary keywords having ONLY in them cannot be specified with any other secondary keywords from this set. Possible choices are from the following subsets:

*acute\_filter*

ACUTE ALSO — Specifies to measure edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees. This is the default behavior if you do not specify a choice from this subset in the operation.

ACUTE ONLY — Specifies to measure only edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same Internal operation.

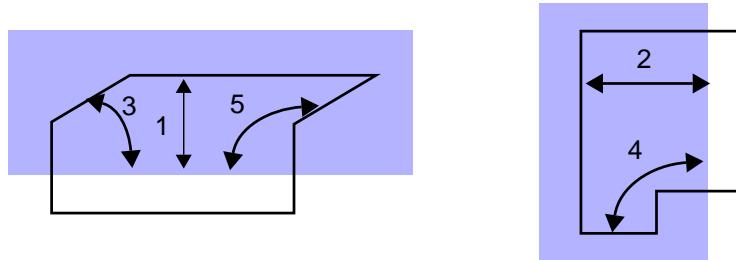
NOT ACUTE — Specifies *not* to measure edges with an appropriate angle  $a$ , such that  $0 < a < 90$  degrees.

Referring to [Figure 4-134](#):

- ACUTE ALSO checks configurations 1 through 3 (this assumes none of the other default settings are changed).
- ACUTE ONLY checks configuration 3.
- NOT ACUTE checks configurations 1 and 2.

Orientations 4 and 5 could be checked if the appropriate filters are set (for example, PERPENDICULAR ALSO and OBTUSE ALSO).

**Figure 4-134. Edge Orientations**



*parallel\_filter*

PARAllel ALSO — Specifies to measure parallel edges in addition to non-parallel edges. This is the default behavior if you do not specify a choice from this subset in the operation.

PARAllel ONLY — Specifies to measure only parallel edges. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same Internal operation.

NOT PARAllel — Specifies *not* to measure parallel edges.

Referring to [Figure 4-134](#):

- PARALLEL ALSO checks configurations 1 through 3 (this assumes none of the other default settings are changed).
- PARALLEL ONLY checks configurations 1 and 2.
- NOT PARALLEL checks configuration 3.

Orientations 4 and 5 could be checked if the appropriate filters are set (for example, PERPENDICULAR ALSO and OBTUSE ALSO).

*perpendicular\_filter*

NOT PERPendicular — Specifies *not* to measure perpendicular edges. This is the default behavior if you do not specify a choice from this subset in the operation.

PERPendicular ONLY — Specifies to measure only perpendicular edges. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same Internal operation.

PERPendicular ALSO — Specifies to measure perpendicular edges in addition to other orientations.

Referring to [Figure 4-134](#):

- NOT PERPENDICULAR checks configurations 1 through 3 (this assumes none of the other default settings are changed).
- PERPENDICULAR ONLY checks configuration 4.
- PERPENDICULAR ALSO checks configurations 1 through 4.

Orientation 5 could be checked if you set OBTUSE ALSO.

*obtuse\_filter*

**NOT OBTUSE** — Specifies *not* to measure edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees. This is the default behavior if you do not specify a choice from this subset in the operation.

**OBTUSE ONLY** — Specifies to measure only edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees. You cannot use this keyword and a keyword from the other *orientation\_filter* subsets in the same Internal operation.

**OBTUSE ALSO** — Specifies to measure edges with an appropriate angle  $a$ , such that  $90 < a < 180$  degrees.

Referring to [Figure 4-134](#):

- NOT OBTUSE checks configurations 1 through 3 (this assumes none of the other default settings are changed).
- OBTUSE ONLY checks configuration 5.
- OBTUSE ALSO checks configurations 1 through 3 and 5.

Orientation 4 could be checked if you set PERPENDICULAR ALSO.

- *projection\_filter*

Secondary keyword set that instructs dimensional check operations to measure the separation between edges based upon their mutual edge projection. Note that the projections of edges onto other edges (if any) are not the output, rather the results of the Internal operation and associated secondary keywords are output if the specified projection exists. See the description of “[“projection\\_filter”](#) on page 518 with the Enclosure statement for the definition of edge projection.

PROjecting [*projection\_length*] — Specifies to measure the separation between two edges only when one edge projects onto the other edge. If *projection\_length* is also specified (it takes the form of a constraint) and the length of any projection conforms to the *projection\_length* constraint then the results are output. The option’s default behavior is *projection\_length* set to  $\geq 0$ .

NOT PROjecting — Specifies to measure the separation between two edges only when neither edge projects onto the other edge.

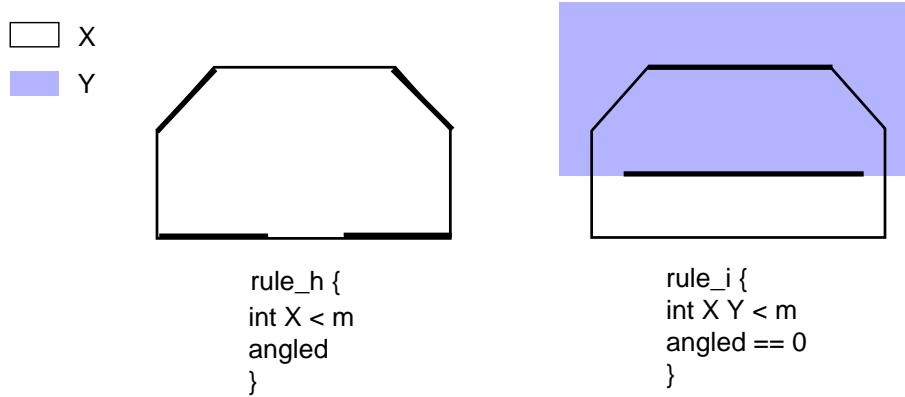
**Hierarchical considerations** — For hierarchical applications, projection and projected length may not be accurately determined if the projection occurs at a hierarchical level well outside the design rule distance between the edges. This problem occurs when two non-parallel edges A and B are being measured with PROJECTING or NOT PROJECTING specified, one or both of the edges extends across multiple hierarchical levels, and the actual projecting portions are well outside the design rule distance between the edges. For constrained PROJECTING filters, if you do not specify the PARALLEL ONLY keyword, Calibre issues a warning and sets the PARALLEL ONLY keyword filter. If you have an unconstrained PROJECTING filter and you want hierarchical and flat results to be the same, it is highly recommended that PARALLEL ONLY be specified.

- *angled\_filter*

Secondary keyword set that instructs dimensional check operations to measure edge pairs based on orthogonality with respect to the coordinate system axes.

**ANGLED [angle\_constraint]** — Specifies to measure the two edges only when the number of non-orthogonal edges in the pair meets the given constraint. The constraint is optional and when omitted, defaults to > 0. See Figure 4-135.

**Figure 4-135. ANGLED**



Examples:

**INT poly < 3 ANGLED == 2 // Measure 2 edges only if both are angled.**  
**INT poly < 3 ANGLED < 2 // Measure 2 edges only if one or neither is angled.**

- *corner\_filter*

Secondary keyword set that instructs dimensional check operations to measure edge clusters based on corner-to-corner or corner-to-edge orientation.

You cannot specify this filter with an orientation, projection, or angled filter.

**CORNER TO CORNER** [*corner\_constraint*] — Specifies to measure the edge clusters only if they are in a corner-to-corner configuration. The constraint is optional. When you specify *corner\_constraint*, the only allowed options are == 45 and != 45. The constraint limits the measurement of edges according to the angle that the line segment, which links the opposing corners, makes with the x-axis.

**CORNER TO EDGE** — Specifies to measure the edge clusters only if they are in a corner-to-edge configuration (defined later).

**CORNER** — Specifies to measure the edge clusters only if they are in a corner-to-corner or corner-to-edge configuration.

**NOT CORNER** — Specifies to measure the edge clusters only if they are not in a corner-to-corner or corner-to-edge configuration.

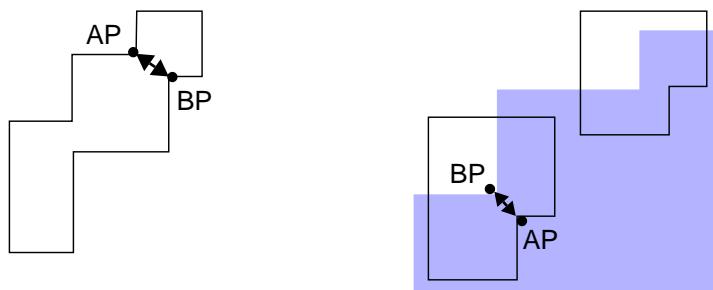
The corner-to-corner and corner-to-edge filters are defined as follows: For a single-layer dimension check, let A and B be two edges from *layer1* to be measured. For a two-layer check, let A be the edge from *layer1* and B the edge from *layer2*. A *convex* 90-degree corner of a polygon is one where the polygon inside covers 90 degrees around the corner point and a *concave* 90-degree corner is one where the polygon inside covers 270 degrees.

The edge clusters are in a CORNER TO CORNER configuration when:

- The edges are in parallel and do not project.
- Let AP be the point on edge A closest to edge B and let BP be the point on edge B closest to edge A. (These points are guaranteed to be unique due to the parallel and projecting restriction.)
- Both AP and BP lie at concave 90-degree corners of polygons on their respective layers. See Figure 4-136.

**Figure 4-136. CORNER TO CORNER**

	X	rule_j {int X < m corner to corner}
	Y	rule_k {int X Y < m corner to corner}



The two edges are in a CORNER TO EDGE configuration when the following are true:

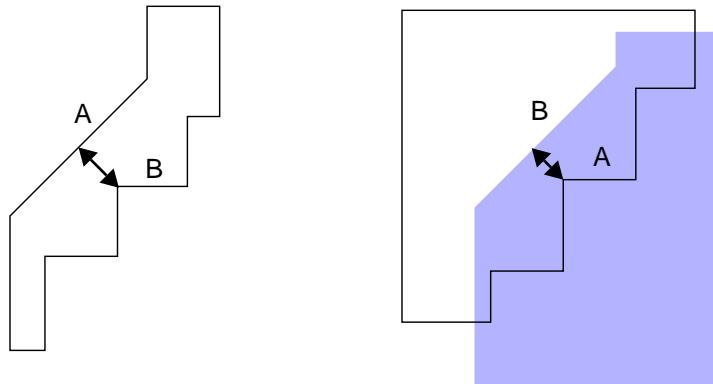
- o The edges are neither parallel or perpendicular.
- o Either endpoint of A or B is at a convex 90-degree corner and the corner projects onto the other edge (that is, the line bisecting the corner intersects the edge at a unique point and this point can be an endpoint of the edge). See Figure 4-137.

**Figure 4-137. CORNER TO EDGE**



rule\_l {int X < m  
corner to edge}

rule\_m {int X Y  
corner to edge}



- *intersection\_filter*

An optional parameter that measures edge pairs based upon intersection behaviors. The value of *intersection\_filter* takes the following form:

[ABUT [*abut\_constraint*]] [OVERLAP] [SINGULAR]

[*intersection\_filter* INTERSECTING ONLY]

You can specify any combination of the secondary keywords ABUT, OVERLAP, and SINGULAR in one operation:

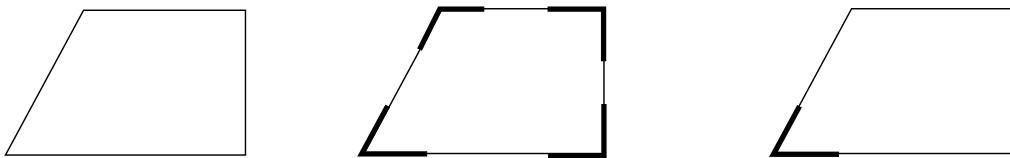
ABUT [*abut\_constraint*] — Specifies that separation between intersecting edges should also be checked. It is highly recommended that you use ABUT < 90 in your rule file unless you have good reasons not to. Intersecting (abutting) edges are not checked by default. Intersecting edges are measured *only if* the appropriate angle between them conforms to the optional *abut\_constraint* (interpreted in degrees). With the exception of intersection, the edges must also meet the criteria of the dimensional check operations and any parameters specified from the *connectivity\_filter*. Output from the ABUT condition is in addition to any other output the Internal operation generates.

The *abut\_constraint* modifier must contain non-negative real numbers less than 180. Single-operator constraints such as < 90 and > 135 are interpreted as  $\geq 0 < 90$  and  $> 135 < 180$ . The default value is  $\geq 0 < 180$ .

If the *abut\_constraint* modifier includes zero in its range (for example, < 90, == 0,  $\geq 0 < 45$ ), then any edges A of layer X and B of layer Y, which are *coincident outside*, are also output (since the angle between the exterior side of A and the interior side of B is zero). There is no measurement involved in this event and polygon containment criteria are not applied.

In a one-layer dimensional check operation, coincident edges do not exist since nmDRC applications automatically merge all abutting or overlapping polygons into one polygon. Therefore, the == 0 and the  $\leq 0$  constraints cannot be specified with the ABUT parameter in single-layer syntax.

This measurement of intersecting edges ignores orientation, projection, and corner filters, but does not ignore connectivity, polygon, and angled filters. See [Figure 4-138](#) for single-layer examples.

**Figure 4-138. ABUT**

rule\_n {int X < m}  
//abutting edges  
//not checked  
//no violations

rule\_o {int X < m ABUT}  
//all abutting edges  
//are checked

rule\_p {int X < m  
ABUT < 90}

**OVERLAP** — Specifies that measurement of the separation between intersecting edges at points where a polygon from one input layer crosses a polygon from the other input layer should also occur. All edges forming the point of overlap are measured as if you specified an unconstrained ABUT parameter. This overrides any specified ABUT parameter but only at the point of overlap. You cannot use this parameter when either of the input layers is a derived edge layer because overlaps cannot be computed in the absence of polygons.

The precise definition of a point at which overlap is detected is as follows:

Given a two-layer dimensional check operation between *layer1* and *layer2*, let P be any point where more than one edge from *layer1* is present and more than one edge from *layer2* is present (after edge-breaking, see the “Edge Breaking” section of the *Calibre Verification User’s Manual* for details). The tool detects an overlap at point P if both the following criteria are true:

- An edge from *layer1* that is outside of or coincident outside with a polygon from *layer2* and an edge from *layer1* that is inside of or coincident inside with a polygon from *layer2*.
- An edge from *layer2* that is outside of or coincident outside with a polygon from *layer1* and an edge from *layer2* that is inside of or coincident inside with a polygon from *layer1*, then an overlap is detected at point P.

**SINGULAR** — Specifies that the separation between the corresponding sides of intersecting edges at points of polygon singularity should also be measured.

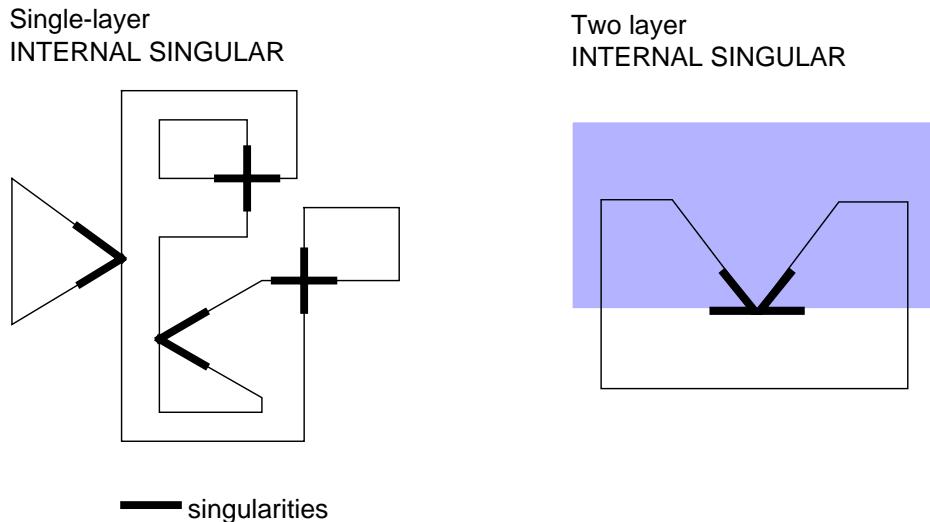
*Singularities* are point-to-edge or point-to-point polygon intersections or self-intersections and are often design rule errors. It is highly recommended that you use SINGULAR in your rule file (unless you have good reasons not to) because singularities are not checked by default. Figure 4-139 shows examples of singularities detected by an Internal operation. Note in the examples, the violations are due to the singularities present. These violations would be output in addition to the usual Internal measurement violations.

When a dimensional check operation detects a point of singularity, all edges forming the point of singularity are measured as if you specified an unconstrained ABUT parameter

in the operation. This overrides any specified ABUT parameter but only at the point of singularity.

You cannot use the SINGULAR parameter when either of the input layers is a derived edge layer because singularities cannot be computed in the absence of polygons.

**Figure 4-139. Internal: Measurement and Output at Singularities**



**INTERSECTING ONLY** — Specifies to measure only between intersecting edges and to ignore non-intersecting edges. Must be preceded with at least one of the other filters in this set. This filter ignores orientation, angled, projection, and corner filters.

Remember, ABUT, OVERLAP, and SINGULAR output their results *in addition to* the typical Internal output. INTERSECTING ONLY limits output to only what the *intersection\_filter* keywords produce. Figures 4-138 and 4-139 show the types of output to expect from INTERSECTING ONLY. Figure 4-127 is not because the measured edges do not intersect.

Here is an example of a common configuration of intersection keywords:

```
// Poly width =2, interior acute angles not ok, point-to-point
// touching of polygons not ok.
poly_width {INT poly <2 ABUT < 90 SINGULAR}
```

- EXCLUDE FALSE

A secondary keyword used to eliminate rare false errors due to the absence of endpoint blocking edges at the correct hierarchical level when two edges are measured. This is often referred to as a false notch. This only applies to hierarchical runs. This keyword should only be used if false errors are actually detected and they cannot be tolerated. This option has a considerable runtime penalty.

See “[False Measurement Reduction](#)” in the *Calibre Verification User’s Manual* for more details.

- *region\_output*

An optional keyword that instructs dimensional check operations to generate a derived edge or polygon layer instead of a derived error layer. (Derived edge and polygon layers can be further manipulated by other operations, whereas derived error layers cannot.) For detailed information on this subject, see “Polygon-Directed Output” in the *Calibre Verification User’s Manual*.

The value of output takes the following form:

`REGION [EXTENTS | CENTERLINE [value]]`

**REGION** — Constructs edge projections between the endpoints of selected edges to create polygonal regions; the composite of the selected edges and edge projections is output as derived polygon data.

If you are deriving polygon layers, using the REGION keyword with the Euclidean (default) metric can generate large numbers of skew edges, which must be flattened in hierarchical applications. This is usually wasteful and requires much excess processing time. REGION EXTENTS is usually the best choice for deriving polygon layers. You can also use the OPPOSITE metric to eliminate skew edges, but be aware that you can miss legitimate design rule errors (such as corner-to-corner configurations) when using OPPOSITE.

Skew output edges resulting from the REGION keyword require special handling. The final placement of such edges is the result of merging and numeric rounding. This can result in the placement of skew edge vertices that differ up to 1 dbu from the layout geometry.

**REGION EXTENTS** — Constructs derived polygon data as for REGION, but the output is the rectangular extents of the polygons output by REGION, rather than the polygons themselves. The tool forms the extents prior to the merging of the regions.

#### Note



We recommend using REGION EXTENTS in most layer derivation applications. You should *not* use REGION EXTENTS when polygonal regions are already orthogonal to the database axes, such as those created with the OPPOSITE metric.

**REGION CENTERLINE [*value*]** — Constructs derived polygon data as for REGION. The output consists of the centerlines of the polygonal regions, rather than the regions themselves. These centerlines are formed prior to the merging of the regions. The centerlines are along the direction of the edges whose measurement forms the region; they have a default width of eight database units. The optional parameter *value* allows you to specify the centerline width. The *value* must be a floating-point number greater than or equal to two database units.

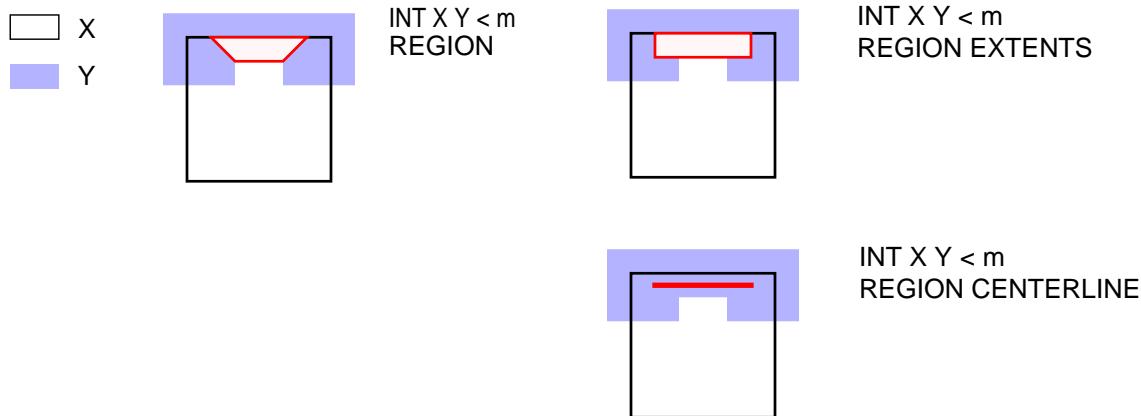
For two, parallel, horizontal edges, the y-coordinate of the centerline segment is always closer to the bottom edge if any snapping occurs. The formula for the y-coordinate of the centerline is:

$$y1 + ( ( y2 - y1 ) / 2 )$$

For vertical edges, a similar change keeps the x-coordinate of the line segment closer to the left edge.

Figure 4-140 shows examples.

**Figure 4-140. REGION Examples**



## Examples

For additional examples see “Width Checks” on page 2073.

## Label Order

Connectivity extraction

**LABEL ORDER** *layer* [...] [DIRECT] [MASK]

### Parameters

- layer

An original layer or layer set, or a derived polygon layer. This *layer* must appear as an input layer to a [Connect](#) or [Sconnect](#) operation in the same verification set. You can specify *layer* any number of times in one statement.

- DIRECT

An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.

- MASK

An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

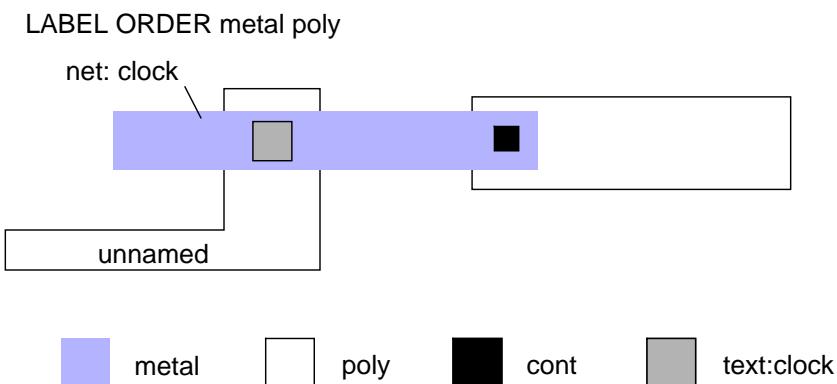
### Description

Defines the order in which connectivity extraction searches layers for objects (typically shapes and paths) that intersect another object owning a label location (or other significant location). The intersecting object whose layer appears first in the Label Order operation is labeled with the value of the net name. See “Label Attachment” in the [Calibre Verification User’s Manual](#).

Net names and port objects are ordered by the Label Order operation. In Calibre applications, this is performed for [Port Layer Text](#) and [Port Layer Polygon](#) objects.

For example, the text shape shown in Figure 4-141 owns a net name with the value of clock and intersects a metal and a poly shape. Because the metal layer appears first in the Label Order operation, connectivity extraction assigns clock as the name of the net with the metal shape. The net with the poly shape that also intersects the text shape is unnamed.

**Figure 4-141. Label Order**



Connectivity extraction uses the Label Order operation only if the layer of the object owning the net name is not specified as an input layer to a Connect operation or is not specified as *layer* to an [Attach](#) operation.

If the input layer(s) of the Label Order operation does not have any objects that intersect the object owning a net name, or if the Label Order operation is not included in a rule file, then the net name is ignored and a warning message is issued.

In ICverify applications, this is performed for Pyxis Layout ports created with the `$make_port()` command. ICverify ports on layers that do not appear in an Label Order operation are reported as unattached.

**Requirements and Restrictions** — You cannot use a Mask-mode Label Order operation in the rule file when [DRC Incremental Connect](#) YES is specified. This situation generates a compilation error.

The following restrictions apply when you use original layer sets as input layers to the connectivity extraction operations:

- A member of the original layer set cannot also be used as an input layer to the operation or as an input layer to another connectivity extraction operation in the same verification set as the operation containing the original layer set.
- Any two *distinct* original layer sets used in one or more connectivity extraction operations in the same verification set (Direct or Mask) cannot have a non-empty intersection (which means the same layer member cannot appear in both distinct original layer sets).
- There can be only one Label Order operation corresponding to each connectivity extraction mode within a rule file.

See also [Text Layer](#), [Layout Text](#), and [Layout Text File](#).

## Examples

```
LABEL ORDER met5 met4 met3 met2 met1 poly
// attach text labels in the layer order specified
```

# Layer

Specification statement

**LAYER** *name* *original\_layer* [*original\_layer* ...]

## Parameters

- ***name***

A required name for an original layer or layer set. Each ***name*** must be a unique string that complies with the restrictions listed under “[General Syntactic Conventions](#)” on page 41. In general, letters, numbers, period (.), and underscore (\_) are allowed in a ***name***.

- ***original\_layer***

A required original layer or layer set. Possible choices are as follows:

- The layer number of an original layer. The allowed range of layer numbers is 0 to 65535.
- The ***name*** of a layer or layer set defined by another Layer operation.
- In Pyxis Layout, the name or alias of an original layer (set) defined in the Pyxis Process.

You can specify ***original\_layer*** any number of times in one statement.

## Description

Defines the name of an original layer or a layer set. Layer statements associate the original layer ***name*** with a set of one or more simple layer numbers. You can reference original layers in your rule file by a ***name*** or number in the case of simple layers, or by ***name*** in the case of layer sets.

When defining the name of a simple layer, the ***original\_layer*** argument must either be a single number, or a single ***name*** defined in another Layer statement. For example:

```
LAYER poly 1 // simple layer
LAYER poly2 poly // simple layer referencing another simple layer
```

A layer set is a group of layers assigned to one ***name***. When defining a layer set, more than one ***original\_layer*** argument is associated with a ***name***. A layer set can be built up from simple layers, other layer sets, or both. For example:

```
LAYER diff 2 3 4 // layer set
LAYER more_diff diff 5 // layer set containing layers 2, 3, 4, and 5
```

Using the same ***name*** for multiple Layer statements is not allowed. Layer ***name*** parameters that contain whitespace or that conflict with SVRF keywords must be enclosed in double quotes (“ ”). You should avoid using SVRF keywords as layer names, as this can cause unnecessary confusion when reading a rule file.

## DATATYPE and TEXTTYPE Attributes

Calibre applications are *insensitive* to DATATYPE and TEXTTYPE attributes.

If you have the following list of layer-datatype pairs:

```
layer 1 datatype 1
layer 1 datatype 2
layer 1 datatype 3
```

Calibre applications map all of these layer-datatype pairs to Calibre layer 1; the datatypes are ignored.

If your design includes DATATYPE and TEXTTYPE attributes, use the [Layer Map](#) statement to map these attributes to layer numbers and use the Layer statement to assign names.

If Calibre cannot resolve a **name** parameter to an original layer number, Calibre issues the compiler error OLS4.

You can assign layer numbers and datatypes to mask layer output using the [DRC Check Map](#) statement.

**ICverify considerations** — In ICverify applications, layer naming is somewhat more complex because, during compilation, the Pyxis Process is also queried for the definition of original layer names referenced in the rule file. Thus, original Layer specification statements are not mandatory in a rule file with use limited to ICverify support as long as the Pyxis Process defines all original layer names referenced in the rule file.

You may define a portion of the original layer set in the rule file and locate the remainder within the Pyxis Process. You may also define original layers in the rule file in terms of original layers defined in the Pyxis Process. If an original layer is defined in both a Layer specification statement and in the Pyxis Process, then the definitions must match; that is, in ICverify, original Layer specification statements cannot redefine original layer names defined in the Pyxis Process.

After processing of all layer parameters is complete, a subset of the layers or layer set may still be unresolved. Each remaining unresolved layer *must* be an original layer or layer set defined in the Pyxis Process. If not, then a compiler error occurs.

One important function of the compilation process is to type-check and determine the data source for layer parameters of operations. In order to do so, these parameters must be resolved as either derived or original layers. Before any layer definitions or operations are compiled, a set of original layer names is determined by processing all original Layer specification statements. This set consists of layers defined by original Layer specification statements *as well as* original layers looked up in the Pyxis Layout process when encountered in original Layer specification statements. At that point, any re-definition of a known original layer name by a layer definition is prohibited.

As mentioned previously, original layer specification statements are optional in rule files used by ICverify. There are, however, several good reasons for defining the entire original layer set in the rule file, even for ICverify use:

- The rule file is more self-contained.
- The same rule file can be used unchanged by both ICverify and Calibre.
- It provides a consistency check against the Pyxis Layout process, especially if the process changes and the rule file does not, or vice-versa.
- You cannot redefine original layers as layer definitions in the rule file if they are defined by original layer specification statements or resolved from the Pyxis Layout process during the processing of original layer specification statements. Other Pyxis Layout process layers may be redefined.

You should note that original layer names looked up in the Pyxis Layout process during compilation of the rule file are resolved into Pyxis Layout layer numbers *at compilation time*. Therefore, if these values become re-associated while the rule file is loaded, then the loaded rule file cannot represent the true state of the Pyxis Layout system and you should reload them.

## LEF/DEF Layer Mapping

For the Calibre xRC tool, defining LEF/DEF layer mapping is possible using the Layer statement. For example, if the following layer exists in your SVRF rule file:

```
LAYER MET1 44
```

To map the LEF/DEF layer M1 to MET1, add the “\_FDI” suffix to the LEF/DEF layer name as follows:

```
LAYER M1_FDI MET1
```

## Examples

### Example 1

The following example defines the original layer configuration within the rule file. Notice how original layer sets are built up within the definitions. DIFF is a layer set consisting of simple layers 2 and 4, M2 consists of simple layers 11 and 18, and MET2 consists of simple layers 11, 13, 15, and 18.

```
LAYER DIFF 2 4      // SOURCE/DRAIN DIFFUSION
LAYER POLY 5        // POLY
LAYER NIMP 6        // N+ IMPLANT MASK
LAYER PIMP 7        // P+ IMPLANT MASK
LAYER CONT 8        // CONTACTS
LAYER MET1 9        // METAL1
LAYER VIA 10        // VIAS
LAYER M2 11 18      // METAL 2 + CLOCKS
LAYER VCC 13         // VCC METAL 2
LAYER VSS 15         // VSS METAL 2
LAYER PAD 60         // PASSIVATION OPENINGS
LAYER PWELL 14       // P-WELL
LAYER MET2 VCC VSS M2 // ALL METAL2
```

### Example 2

Layer is often used with Layer Map, because Calibre is insensitive to DATATYPES and TEXTTYPES. Here is an example.

```
LAYER MAP 1 DATATYPE > 0 5
// map all layer 1 objects having datatypes > 0 to layer 5

LAYER one 1 // layer 1 datatype 0 is read in on this layer
LAYER poly 5 // all layout data from layer 5 and from the LAYER MAP
// statement is read in on this layer
```

### Example 3

The next example shows how the Layer operation can eliminate a sequence of Boolean operations you might use to construct a layer. First, assume the following:

```
LAYER VSS 15      // VSS metal_2
LAYER VCC 16      // VCC metal_2
LAYER clock 18    // clock metal_2
LAYER M2 17 19    // All other metal_2
```

Then you could use this Layer statement:

```
LAYER met2 VSS VCC clock M2
```

instead of these Boolean operations:

```
X = VSS OR VCC
Y = X OR clock
met2 = Y OR M2
```

# Layer Directory

Specification statement

## LAYER DIRECTORY *filename*

Not used in nmLVS or Calibre xRC applications.

### Parameters

- *filename*

A required filename of a directory for layer data storage.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

### Description

Specifies a directory for disk-based layer storage to be used by the verification application. This statement applies to Calibre tools with the exception of nmLVS and xRC, which do not use layer directories. See the “[Disk-Based Layers](#)” section of the *Calibre Verification User’s Manual* for additional information.

By default, layers are based in memory only. Layer Directory should only be used when there are memory constraints on the host machine that require writing data to disk. Use of this statement *does not* prevent the Calibre tool from using swap space.

If this statement is specified, then disk-based layers are used instead of memory. The disk partition that the *filename* references should have more storage than the maximum LVHEAP value for the run. The *filename* you specify should be on the machine where Calibre is running or the run can become very slow.

The tail of the specified *filename* is created as a directory if it does not already exist (all of the subdirectories in the filename are assumed to exist, except the tail). A subdirectory is created in the specified *filename* to hold the layer files. A test open is performed to determine the integrity of the specified directory. If directory creation fails, or if the test open fails, the program stops by default. These behaviors can be modified by changing the [Layout Input Exception Severity](#) LAYER\_DIRECTORY setting.

All files created by this statement are deleted when the run is complete.

The DISK value that appears in the run transcript is the size (in megabytes) of the directory created by this statement.

### Flat Mode

The tool writes a subdirectory called icv.<number> at the location specified by the *filename*. Files that hold disk-based layers are named L<number>, where <number> is an internally-generated layer number. The files are created and deleted using the same scheduling algorithms as for virtual-memory-based layers.

## **Layer Directory**

---

In Pyxis Layout, sets the initial default value for the layerdirectory switch in the \$check\_drc() function. The default is the current directory or the filename specified in the Layer Directory statement.

### **Hierarchical Mode**

A hidden file is created at the location indicated by the *filename*. This file stores all information during the run.

For MTflex runs, this statement applies to the master machine only. For information about using layer directories on remote hosts in MTflex mode, see the [REMOTE HOST](#) command in the *Calibre Administrator's Guide*.

### **Examples**

The following statement creates the “layers” directory; however the directories above that level must already exist:

```
LAYER DIRECTORY "/tmp/ring.cmos/layers"
```

## Layer Map

Specification statement

```
LAYER MAP source_layer {DATATYPE source_type target_layer
| TEXTTYPE source_type target_layer [target_texttype]}
```

Used only in Calibre.

### Parameters

- ***source\_layer***

A required positive integer, or one of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

An integer specifies a layer in the layout database, whereas a constraint specifies a range of layers in the layout database.

- ****DATATYPE****

Keyword that instructs the tool to create a DATATYPE map. This is used to map drawn layers.

- ****TEXTTYPE****

Keyword that instructs the tool to create a TEXTTYPE map. This is used to map text objects.

- ***source\_type***

A required positive integer, or one of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

The integer specifies a particular datatype or texttype in the layout database, whereas a constraint specifies a range of types.

- ***target\_layer***

A required positive integer that specifies the layer number to be used by Calibre. The range of possible layers is 0 to 65535.

- ***target\_texttype***

An optional integer that specifies the TEXTTYPE attribute to map the ***source\_type*** to. May only be used with the **TEXTTYPE** keyword. The range of possible values is 0 to 65535.

### Description

Enables Calibre to map layer numbers, DATATYPE attributes, and TEXTTYPE attributes in GDS and OASIS databases to layer numbers that Calibre uses in the rule file. Layer Map is used in conjunction with the [Layer](#) statement for establishing *simple layer* number assignments in the

rule file. A simple layer number is the primitive number of a layer after all Layer Map statements have been applied.

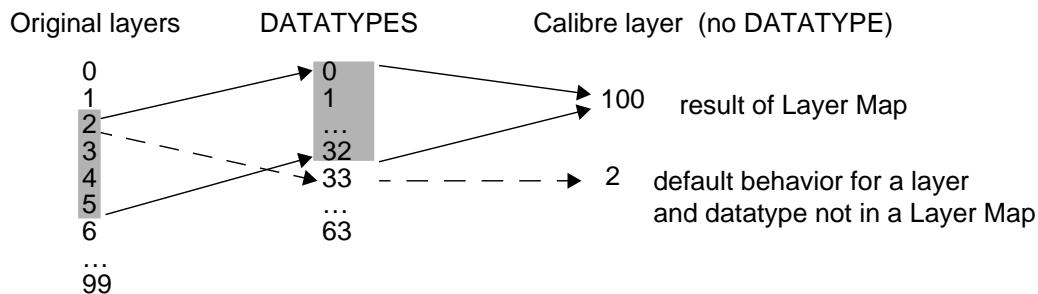
Calibre applications are *insensitive* to DATATYPE and TEXTTYPE attributes. For example, if you have the following list of layer datatypes:

```
layer 1 datatype 1  
layer 1 datatype 2  
layer 1 datatype 3
```

Calibre applications map all of these to Calibre layer 1 if no Layer Map statement applies to them. Notice that Calibre uses no DATATYPE for the internal representation of these layers. Handling of TEXTTYPE attributes is similar.

For datatype maps, the layout reader proceeds as described next (texttype maps are similar and apply to text objects rather than geometry). Consider this layer map:

```
LAYER MAP >= 2 <= 5 DATATYPE < 33 100
```



This causes any polygon P on layers 2 through 5, and having DATATYPE less than 33, to be treated as Calibre layer 100, because P's layer and datatype match the ranges for *source\_layer* and *source\_type*, respectively. However, if P is on layer 2 and has datatype 33, then it is mapped to Calibre layer 2.

In the preceding example, if you have data on original layer 100 in your layout database and your rule file causes layer 100 to be read in from the database, that data appears on Calibre layer 100 *in addition to* the data placed on Calibre layer 100 by the Layer Map statement.

The same layer and datatype pairs can be mapped to multiple layers. For example, assume a second Layer Map statement is specified in addition to the Layer Map statement shown in the previous example:

```
LAYER MAP >=2 <=5 DATATYPE < 33 1  
LAYER MAP < 5 DATATYPE < 26 5 // no layer reassignment occurs here
```

The *source\_layer* and *source\_type* of the second Layer Map statement are a subset of the source layers and datatypes specified in the first statement. Regardless, the statements act independently on the specified original layers and the second statement does not reassign the results of the first statement to a new target layer. In this example, the layers and datatypes that are common to both of these statements would be assigned to both layers 1 and 5 in memory.

Note that datatype maps are completely different from layer sets. The Layer statement names pertain only to shapes read from simple layers *after any layer mapping has occurred*. For example:

```
LAYER POLY 1
```

Here, POLY is the name of a simple layer associated with Calibre layer 1. This is in contrast to a layer set, which may appear like this:

```
LAYER POLY 1 2 3 // layer set
LAYER MORE_POLY POLY 4 // layer set containing layers 1, 2, 3, and 4.
```

The 1-to-N layer name association for a *layer set* occurs after all simple layers have been read by the layout reader. In the preceding Layer statements, this means layers 1 through 4 undergo any layer mapping from Layer Map statements *before* the layers named POLY and MORE\_POLY are established as layer sets.

Unexpected warnings can occur if data from a **target\_layer** is not mapped. For example, if the layout contains layer/datatype combinations of (1, 2) and (28, 4), consider these statements:

```
LAYER pwell 28
LAYER MAP 1 DATATYPE 2 28
```

In this case, layer (1, 2) and all current datatypes on layer 28 are mapped to layer pwell (Calibre layer 28). However, Calibre can issue this warning in the run transcript and the nmDRC summary report (if specified):

```
Layer 28 contains unmapped objects and is the target layer of LAYER MAP 1
DATATYPE 2
```

When this warning is issued, layer 28 is the target of a layer mapping but layer 28 has datatype records already associated with it that have not been explicitly mapped. The warning is intended to draw attention to the (28,4) data that was originally on layer 28 and will remain on layer 28 in addition to the (1, 2) data that is mapped to layer 28.

This warning occurs when the following conditions are true for an input layer:

- It is the **target\_layer** of a datatype map is required by the flow.
- It contains objects which themselves are not mapped by any datatype map.

The run is allowed to proceed by default. You can change this behavior to cause a fatal error by specifying:

```
LAYOUT INPUT EXCEPTION SEVERITY DATATYPE_MAP_TARGET 2
```

To avoid this situation, you could map intervals of layers to a single Calibre layer (that is intended to be ignored during the run) first, and then remap all of the layers that have the desired data on them to new Calibre layers. Doing so does not introduce any performance issues, as Calibre only reads in layers actually required for the run. See Example 2 in the Examples section.

A similar warning as discussed previously occurs if some, but not all, data is mapped from the source layer. By default, Calibre issues a warning if there are unmapped objects on a

*source\_layer*. You can control the severity of this situation with the [Layout Input Exception Severity](#) DATATYPE\_MAP\_SOURCE setting.

TEXTTYPE maps apply to text objects that exist on [Layout Property Text](#), [Layout Text](#), [Layout Text File](#), [Text Layer](#), [Port Layer Text](#), and [Device](#) TEXT MODEL LAYER and TEXT PROPERTY LAYER parameters in a similar way to DATATYPE maps for datatypes and original layers. Any texttype attribute of a text object is *retained throughout the Calibre flow* by default. For example, text objects output using [DRC Map Text YES](#) include a texttype value.

If a TEXTTYPE mapping is specified but *target\_texttype* is not, then the texttype is that of the original object regardless of the fact the text object has been mapped to a new layer.

TEXTTYPE maps (not DATATYPE maps) allow the GDSII or OASIS texttype of a text object to be changed by supplying a *target\_texttype* parameter.

If you are using DRC Map Text YES with [DRC Map Text Layer](#), the Layer Map *target\_layer* parameter is *not used* by the DRC Map Text Layer statement. You must use the original layer number from the input design in the DRC Map Text Layer statement because that is where the original text objects reside. Any subsequent layer mappings of original layer numbers will cause text objects to be mapped to the *target\_layer*.

The section “Datatypes and Texttypes in Calibre” in the [Calibre Verification User’s Manual](#) has a detailed discussion of layer mapping.

You can assign layer numbers and datatypes to mask data output from Calibre nmDRC by using the [DRC Check Map](#) statement.

## Examples

### Example 1

This example shows a typical layer mapping.

```
LAYER diff      1
LAYER diffnsb  108

LAYER MAP 1 DATATYPE >= 7 <= 32 108
// Map layer 1 datatypes between 7 and 32 to layer 108.
// Layer diffnsb is an alias for layer 108.
// All other layer 1 datatypes are mapped to diff.
```

If there is any original layer data on layer 108, it is read in on layer 108 in addition to any data from the Layer Map statement.

### Example 2

This example shows how to avoid Layer Map warnings for unmapped objects read in from the layout.

```
LAYER IGNORE 999
// be certain no layout data is read from layer 999, or this will
// cause a warning

// first, map everything to the IGNORE layer
LAYER MAP >= 0 DATATYPE >= 0 999
```

```
// now map all the layers you in which you are truly interested
LAYER MAP 1 DATATYPE 0 10
LAYER MAP 1 DATATYPE 1 11
...
LAYER nwell 10
LAYER nwroute 11
...
```

### Example 3

Suppose you have the following statements in your rule file:

```
LAYER prop_text 10
...
DEV MP ... TEXT PROPERTY LAYER prop_text [ ...
```

Suppose that the prop\_text layer is on layer 50 texttype 10 in the layout database and you want to map the property text objects for your Device statement to Calibre layer 10 (texttype remains 10), as shown. To map these objects in this way, you can do the following:

```
LAYER MAP 50 TEXTTYPE==10 10
```

# Layer Resolution

Specification statement

**LAYER RESOLUTION** *layer* [*layer* ...] {*s* | *x* *y*}

## Parameters

- *layer*

Required names (not numbers) of any number of original layers.

- *s* | *x* *y*

A required positive integer, or pair of positive integers, that specifies the layout grid step size in database units. Specifying one value, *s*, causes the grid step to be square (equal in x- and y-directions). Specifying *x* *y* indicates that *x* is the x-direction grid step and *y* is the y-direction grid step. The default behavior is *s* equal to one database unit, if you do not include this statement in the rule file.

## Description

Overrides the default [Resolution](#) specification statement parameters for any number of original layers during off-grid vertex checking for [Flag Offgrid](#), [Drawn Offgrid](#), [Offgrid](#), [Snap Offgrid](#), or the ICrules CHEck DRc -FLAGOFFGRID option. The *layer* parameters must each be layer names specified in a [Layer](#) statement.

During off-grid checking, all shapes on each *layer* are checked using the resolution specified in the Layer Resolution statement rather than the default specified in the Resolution specification statement. As with all drawn geometry flagging, each *layer* must be required by the nmDRC check set (that is, each *layer* must be required by the flow in order to be read in). Do not use layer set names for the *layer* parameters if your intent is simply to override the default resolution for the constituent layers of the layer set.

The statement may be specified any number of times (for any number of layers), but only once for each original layer.

See also [Precision](#).

## Examples

```
PRECISION 1000
RESOLUTION 100
LAYER RESOLUTION poly1 poly2 50 // align on a .10 um grid
                                // align poly1 and poly2 on a .05 um grid
FLAG OFFGRID YES              // all geometry must be on .10 um grid
                                // except poly must be on .05 um grid
```

# Layout Allow Duplicate Cell

Specification statement

## LAYOUT ALLOW DUPLICATE CELL[S] {NO | YES}

Used only in Calibre.

### Parameters

- **NO**

Keyword that instructs the tool not to allow multiple records for the same layout cell. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to allow multiple records for the same layout cell.

### Description

Specifies whether multiple records for the same layout cell are allowed for the geometric input layout database.

By default, all cell records with the same name after the first are discarded with a warning or error message.

If **YES** is specified, then multiple cell records are treated as if the data were combined into a single record with no warning or error message. This can be useful when the database is split into multiple files by layer.

The **YES** option is equivalent to [Layout Input Exception Severity DUPLICATE\\_CELL 3](#). If the DUPLICATE\_CELL exception is explicitly set in a Layout Input Exception Severity statement, then that setting has priority.

See also [Layout Rename Cell](#), [Layout Path](#), and [Layout Path2](#).

### Examples

```
// Do not allow duplicate cell names in the input stream.  
LAYOUT ALLOW DUPLICATE CELL NO
```

## Layout Base Cell

Specification statement

**LAYOUT BASE CELL** *name* [*name* ...]

Used only in hierarchical Calibre.

### Parameters

- *name*

A required name of a base cell for gate array designs. You can specify *name* any number of times in one statement. The *name* can be a string variable (see [Variable](#)). The *name* parameter can contain one or more asterisk (\*) wildcard characters, where the \* character matches zero or more characters. When using the wildcard, enclose the cell name in quotation marks. Otherwise, a compilation error occurs because the asterisk is a reserved symbol. If you use just an \* character, the top cell is also included in the statement.

### Description

Specifies base cell names for gate array designs. Calibre attempts to identify base cells automatically in gate array or embedded array designs, but errors are possible. Providing base cell names explicitly with this statement provides considerable performance improvements, although doing so has the disadvantage of making the rule file design-specific. Specifying this statement in a header file is, therefore, a preferred practice.

Guidelines for using this statement:

- Base cells should be relatively simple and have little sub-hierarchy.
- Base cells should predominate the placements in the design.
- Base cells should not be contained in the hierarchy of other base cells.
- Base cells should not be used if the design contains embedded memory.

This statement can be specified any number of times.

### Examples

```
// Specify the base cell names.  
LAYOUT BASE CELL cella "base_*
```

# Layout Base Layer

Specification statement

**LAYOUT BASE LAYER** *layer* [*layer* ...]

Used only in hierarchical Calibre.

## Parameters

- *layer*

A required name of an original (drawn) layer. You can specify *layer* any number of times in one statement.

## Description

### Note



Use of this statement or [Layout Top Layer](#) is strongly recommended for all hierarchical applications. Layout Base Layer is usually preferable from a rule file maintenance standpoint.

Specifies device-level layers for performance tuning of hierarchical applications. The hierarchical database constructor creates an internal model of the design hierarchy that is optimal for runtime performance. The primary optimization that this statement provides is the extremely important ability for Calibre to distinguish cells whose primary function is routing from cells that comprise the *device plane* of the design. The former are often called *top-layer cells*.

Layout Base Layer enumerates a set of original (drawn) layer names that are considered to be device-level layers (or base layers) and are not found in top-layer cells. These base layers are generally distinct from top-level layers. Cells that contain only top-level layers as original data (including through the sub-hierarchy of the cell) are identified as top-layer cells and are subjected to a host of internal optimizations.

Typically, base layers contain polygon configurations that are likely to be replicated elsewhere on the mask. Recommended layers to include are device-forming layers like poly and diffusion. Implant layers and contact (not via) layers are also good candidates. Well and substrate layers should not be specified, even though these layers can be used for device bulk pins. Metal, via (not contact), solder bump, pad, fuse, and special-purpose layers such as markers, artificial cell boundaries, and text should not be used in this statement.

In some cases, a metal layer can be used in a device-level (seed) layer within cells, but not for interconnect routing between cells. In these cases, the metal layer used in the device-level layer must be declared in a Layout Base Layer statement. If a metal layer is used both in the device-level layer and to connect cells, then do not specify that layer in a Layout Base Layer statement.

This statement should be used when the design incorporates high-level power rails or bumps as placements instead of polygons. Typically, these placements cover the entire design, which can cause the system heuristics problems in distinguishing such a design from a gate array.

## Layout Base Layer

---

This statement is a counterpart to Layout Top Layer and is easier to maintain than Layout Top Layer for multiple-process rule files. This is because base layers are unlikely to change when new metal layers are added to a process.

Specifying Layout Base Layer invokes the same optimization heuristics as [Layout Top Layer](#) for layers that are considered top layers. A layer L is a top layer if either of the following is true:

- A Layout Base Layer statement is specified and L is not in a Layout Base Layer statement.
- No Layout Base Layer statement is specified and L is in a Layout Top Layer statement.

You can specify this statement any number of times and use it only with hierarchical Calibre applications. In nmLVS-H, specified top-layer cells are expanded by one level.

**Top-layer cell recognition** — When Layout Base Layer is specified, the layers that are in the layout database and that are not defined as base layers are used for top-layer cell recognition. This allows consistent top-layer cell recognition even if only a subset of the flow is used (which can cause only a subset of the database layers to be read in for the run).

**Cell placement pushdown optimization** — If either Layout Base Layer or Layout Top Layer is properly specified, then only top-layer cell placements are candidates for pushdown, and non-top-layer cell placements are never pushed down. Top-layer cell placements are placements of cells that do not contain any layers other than those that are considered top-level, as defined previously. Cell pushdown is an optimization in the hierarchical database constructor that, under certain conditions, pushes cell placements down into underlying cells.

If neither Layout Base Layer nor Layout Top Layer are specified, then all layers are treated as base layers, and the usual criteria are used to select candidates for pushdown (specifically, very small cell placements are candidates for pushdown). When Layout Base Layer or Layout Top Layer is properly specified, the hierarchical database constructor in Calibre nmLVS-H does not push cells that contain devices down into other cells. This prevents undesired pushdown of devices into hcells.

## Examples

```
// Specify the base (device-forming) layers.  
LAYOUT BASE LAYER poly diff contact nplus pplus
```

# Layout Bump2

Specification statement

## LAYOUT BUMP2 *value*

Used only in Calibre.

### Parameters

- *value*

A required positive integer that specifies an incremental value to use when defining layers in the second database.

### Description

Specifies that the layer numbers in the second database (when using dual database applications) are increased by the specified *value*. Dual database applications are described in the “[Dual Database Comparison](#)” section in the *Calibre Verification User’s Manual*.

If you specify Layout Bump2, then you must also specify [Layout Path2](#), [Layout Primary2](#), and [Layout System2](#).

When you perform dual database applications, primitive objects need to be distinguished between the two databases. You do this by incrementing the *primitive layer number* (before any [Layer Map](#) statements are applied) of all the objects in the second database by the *value*.

The parameter *value* should be greater than the largest layer number in the first database. All objects in the first database whose primitive layer number is greater than or equal to the *value* are ignored.

The LAYER\_OVER\_BUMP exception of the [Layout Input Exception Severity](#) specification statement controls the handling of objects from the first input layout database whose primitive layer number is greater than or equal to the Layout Bump2 specification statement *value*.

This statement also applies to text objects, including those defined in [Layout Text](#) statements. This statement may be specified once in your rule file.

### Examples

#### Example 1

This example shows a simple layer bump.

```
//SPECIFY FIRST DATABASE
LAYOUT PATH "/tmp/ring_new.gds"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM GDSII

//SPECIFY SECOND DATABASE
LAYOUT PATH2 "/tmp/ring.gds"
LAYOUT PRIMARY2 "TOP"
LAYOUT SYSTEM2 GDSII
LAYOUT BUMP2 200 // add 200 to layer numbers for this database

...
```

```
compare_1 { 1 XOR 201 }
compare_2 { 2 XOR 201 }
```

### Example 2

Layer bump with layer mapping.

```
LAYOUT BUMP2 500

//LAYER MAPPING

//DATABASE 1
  LAYER MAP 1 DATATYPE 40 2
    //LAYER 1 DATATYPE 40 MAPPED TO LAYER 2
    LAYER ACT_EX 2
      //ACT_EX IN DATABASE 1 ASSIGNED TO LAYER 2

//DATABASE 2
  LAYER MAP 501 DATATYPE 40 502
    //LAYER 501 DATATYPE 40 MAPPED TO LAYER 502
    LAYER ACT_EX2 502
      //ACT_EX2 ASSIGNED TO LAYER 502
```

## Layout Case

Specification statement

### LAYOUT CASE {NO | YES}

Used only in Calibre nmLVS-H and xRC.

#### Parameters

- **NO**  
Keyword that instructs the tool not to process the layout netlist (SPICE only) in a case-sensitive manner. This is the default behavior when you do not include this statement in the rule file.
- **YES**  
Keyword that instructs the tool to process the layout netlist (SPICE only) in a case-sensitive manner.

#### Description

Specifies case-sensitive parsing of SPICE netlists. This statement controls case sensitivity in the SPICE reader module but not during LVS comparison. Use [LVS Compare Case](#) to control case-sensitive comparison.

To treat geometric databases in a case-sensitive manner use [Layout Preserve Case](#).

You should specify Layout Case YES if the intent of the design is to have layout netlist elements be treated as different elements if text cases differ. For example, if VDD and vdd should be treated as separate nets, then specifying YES is desirable.

This statement controls the treatment of node names, subcircuit names, model names, and user-defined parameter names such as subcircuit definitions, subcircuit calls, and in .PARAM statements. If such objects differ in text case, then Layout Case YES causes them to be treated as separate objects. Keywords such as built-in element names (M, C, X), dot commands (.SUBCKT), and built-in parameter names (W, L, \$SUB) are not affected.

During netlist extraction, Layout Case checks for duplicate element names during generation of primitive .SUBCKT definitions for user-defined devices in the hierarchical SPICE netlister.

- If Layout Case YES is specified, then two [Device](#) operations with the same element name and number of pins, but different cases for the element name, generate two distinct .SUBCKT definitions in the extracted layout netlist, and the case of the element name is preserved.
- If Layout Case NO is specified (or the statement is omitted) then the element names are considered identical and one common .SUBCKT definition is generated; the case of the element name is arbitrary (usually controlled by the first Device operation, but this is not guaranteed). This is done to avoid duplicate .SUBCKT definitions in the extracted layout netlist.

## Layout Case

---

This statement has no effect during connectivity extraction apart from that described in the preceding paragraphs. Net name comparisons in the connectivity extractor are case-insensitive.

The [Net](#) and [Not Net](#) operations are not affected by the Layout Case statement, but they are affected by Layout Preserve Case with the NET option.

For related information, see “[Selection of Extracted Netlist Names](#)” and “[Case-Sensitive Handling of Netlists](#)” in the *Calibre Verification User’s Manual*.

See also [Source Case](#).

## Examples

Suppose you have the following layout netlist:

```
.SUBCKT block A B Y
X1 net1 B GND gnd cell1
...
.ENDS
```

If the intent of the design is to have GND and gnd be considered the same net, then you should specify Layout Case NO.

If the intent of the design is to have GND and gnd be separate nets, then you should specify Layout Case YES. If this is the intent, then it is frequently desirable that Source Case YES be specified also. You will frequently want to use LVS Compare Case YES or NAMES in these circumstances.

# Layout Cell List

Specification statement

**LAYOUT CELL LIST** *name pattern* [*pattern ...*] [TEXTED]

## Parameters

- ***name***

A required cell list name. Cell list names are case-insensitive, can contain only letters, numbers, and underscores (\_), and must be unique (no two lists can have the same name). The first character of a cell list name must be a letter.

- ***pattern***

A required name pattern that matches zero or more cell names. Matching can be specified with “\*” wildcards, with regular expressions, or list patterns. This parameter may be specified more than once.

- TEXTED

An optional keyword that constrains the list of selected cells to only the cells that contain text objects.

## Description

Used to specify cell lists for [Layout Preserve Cell List](#), [Layout Rename Text](#), [Layout Ignore Text](#), [DFM Copy](#), and [DFM Analyze](#). The ***name*** parameter is the name of a list referenced in a CELL LIST or RESULT CELLS option.

A ***pattern*** can be one of two fundamental types: *inclusive* or *exclusive*. An inclusive pattern indicates that all cells matching this pattern are included in the list. An exclusive pattern indicates that all cells matching this pattern are not included in the list. A pattern is considered exclusive if it starts with the ‘!’ character. The ‘!’ character is not considered to be a part of the pattern.

- Wildcard matching

A ***pattern*** may be described with a cell name and one or more optional ‘\*’ characters, where each ‘\*’ character is a wildcard that matches zero or more characters. Any ***pattern*** using the ‘\*’ character must be enclosed in quotes, otherwise a compilation error will result since ‘\*’ is a reserved symbol. Cell name matching using this method is case-sensitive.

Examples:

“ABC”	includes only one cell name, “ABC”
“AB*”	includes any cell whose name starts with AB
“A*C”	includes any cell whose name starts with A and ends with C
“*”	includes any cell name
“!AB*”	excludes any cell whose name starts with AB

- Regular expression matching

The **pattern** may be a regular expression. The Layout Cell List statement uses the GNU implementation of regular expressions, similar to the Layout Rename Text specification statement. (*Regular expression* is often shorted to *regex* in programming literature and this term is used in this section.) For a discussion of GNU regex implementation, see:

<http://www.gnu.org/software/gawk/manual/gawk.html#Regexp>

A regex pattern is indicated by the leading ‘/’ character, followed by a regex (the ‘/’ character itself is not a part of the regex). The expression may be terminated with another ‘/’ character, but this is not required unless regex flags are used, which are discussed in the following paragraph. A regex pattern must be enclosed in quotes since ‘/’ is a reserved symbol. Exclusive regex patterns begin with a ‘!’ character followed by the ‘/’ character.

By default, regex matching is case-insensitive and uses extended regex syntax. You can change this by specifying flags after the trailing ‘/’ terminating the regex. All flags are optional. Multiple flags can be specified, but each flag can be specified only once. Blank spaces or any other characters not recognized as flags cannot appear in this field. The following flags are recognized:

- ‘e’ — use extended regular expression syntax for this command. This is the default. This flag cannot be specified together with ‘b’.
- ‘b’ — use basic regular expression syntax for this command. This flag cannot be specified together with ‘e’.
- ‘i’ — ignore case when matching a regular expression in this command. This is the default. This flag cannot be specified together with ‘m’.
- ‘m’ — match case when matching a regular expression in this command. This flag cannot be specified together with ‘i’.

Examples of regex patterns (again, case is ignored by default):

“/ABC”	includes only one cell name, “ABC”
“/AB.*”	includes any cell whose name starts with AB
“/A[A-C]/*”	includes cells AA, AB, and AC
“/A[A-C]/m”	includes cells AA, AB, and AC, matching case
“/.*”	includes any cell
“!/AB.*”	excludes any cell whose name starts with AB

Note that a regex pattern requires a full match, not a substring match. For example, pattern “/A.\*” matches only cell names beginning with “A” (in regular expression terminology, the patterns are considered *anchored*).

- List matching

A list pattern contains a reference to another cell list specified by a different Layout Cell List specification statement. Such a reference is indicated by the leading ‘&’ character,

followed by a list name (the ‘&’ character itself is not a part of the list name). A list pattern must be enclosed in quotes since ‘&’ is a reserved symbol. Exclusive list patterns begin with a ‘!’ character followed by the ‘&’ character.

List pattern matching is case-insensitive.

Examples:

“&LIST1” includes all cells in “LIST1”

“!&LIST2” excludes all cells in “LIST2”

Circular list references are not allowed. For example, the following statement is invalid because the list is referencing itself:

```
LAYOUT CELL LIST LIST1 "&LIST1" // error, self-reference
```

The following two statements are also invalid, in combination:

```
LAYOUT CELL LIST LIST1 "&LIST2"
LAYOUT CELL LIST LIST2 "&LIST1" // error, circular reference
```

## Usage Notes

The first **pattern** in a list must be an inclusive pattern because an exclusive pattern does not mean “all cells except the ones matching this pattern are included in the list.” It means “all cells matching this pattern are not included in this list.” For example, consider these lists:

```
LAYOUT CELL LIST LIST1 " !A*" // error
LAYOUT CELL LIST LIST2 "*" " !A*" // correct
```

The first list *does not* mean “include all cells with names not beginning with an A.” The exclusive pattern means that all cells whose names begin with an A are not in the list, but it does not imply that all other cells are in the list. Because there are no inclusive patterns, the first list contains no cells at all. Calibre considers such lists invalid and generates a compilation error if Calibre encounters them. The second list is the correct form to exclude all cells beginning with an A.

A Layout Cell List statement may contain several patterns. Patterns that are more general should precede patterns that are more specific. If a more general pattern follows a more specific one, the more specific pattern is ineffective. For example, in this pattern:

“AB\*” “!A\*”

the “AB\*” pattern is ineffective because “!A\*” excludes every cell name that begins with A. Subsequent patterns modify the patterns specified previous to them. For example, in this pattern:

“\*” “!A\*”

the “\*” includes all cell names, but the “!A\*” modifies the first pattern by excluding all cell names that begin with A.

## Layout Cell List

---

A good way to think about pattern matching is to think of the patterns being matched from right to left. For example, in the following list:

```
LAYOUT CELL LIST LIST1 "*" "!A*" "AB*" "!ABC"
```

beginning from the rightmost pattern and working to the left, the patterns do the following:

“!ABC” — if a cell is named “ABC”, then it is excluded.

“AB\*\*” — if a cell name begins with “AB”, then it is included (“ABC” remains excluded).

“!A\*\*” — if a cell name begins with “A” and has not already been included by the pattern “AB\*\*”, then it is excluded.

“\*” — any other cell name that has not already been excluded is included in the list again, the first pattern in the list must be inclusive and no more specific than the other patterns).

Using the preceding example, cell “AAA” does not match the last pattern or the one before that, but it matches the third pattern from the right, “!A\*”. Since this pattern is exclusive, the cell is not in the list. Cell “BCD” does not match any of the patterns except the fourth pattern from the right, and, since the pattern “\*” is inclusive, this cell is in the list.

## Examples

### Example 1

```
// LIST1 contains all cells whose names begin with "A" or "B".  
LAYOUT CELL LIST LIST1 "A*" "B*"
```

### Example 2

```
// LIST2 contains all cells whose names begin with "A" or "B".  
LAYOUT CELL LIST LIST2 "/[AB].*"
```

### Example 3

```
// LIST3 contains all cells not included in LIST2.  
LAYOUT CELL LIST LIST3 "*" "!&LIST2"
```

### Example 4

```
// LIST4 contains all cells whose names end with a number, except cells  
// included in LIST2.  
LAYOUT CELL LIST LIST4 ".*[0-9]" "!&LIST2"
```

### Example 5

The texted\_cells list will include cells that either contain text objects and match the pattern “pads\_\*”, or contain text objects and are members of list1.

```
LAYOUT CELL LIST list1 "mos_"  
LAYOUT CELL LIST texted_cells "pads_<*>" "&list1" TEXTED  
LAYOUT PRESERVE CELL LIST texted_cells
```

## Layout Cell Match Rule

Specification statement

**LAYOUT CELL MATCH RULE** *rule\_name layer\_rule [layer\_rule ...]*

Used only in hierarchical Calibre tools.

### Parameters

- ***rule\_name***

A required name of a cell matching rule.

- ***layer\_rule***

A required parameter set that specifies a layer handling rule. More than one of these parameter sets may be specified. The form a ***layer\_rule*** is this:

**BY LAYER** *layer1* [INSIDE OF LAYER *layer2*]

- **BY LAYER** *layer1*

Specifies an original layer that is used to determine if cells are geometrically equal.

- INSIDE OF LAYER *layer2*

An optional keyword set that specifies the ***layer1*** parameter is inside of the ***layer2*** parameter in order for the ***layer1*** parameter to be used.

### Description

Specifies “golden” cell matching rules for corresponding [Extent Cell](#), [Inside Cell](#), or [Not Inside Cell](#) operations that specify WITH MATCH RULE *rule\_name* keyword sets. The cell matching rules specified in Layout Cell Match Rule statements override the usual definition of *geometrically equal* cells for golden cell matching purposes. This statement may be specified multiple times.

The ***rule\_name*** parameters that match between the layer operations discussed in the preceding paragraph and Layout Cell Match Rule statements are used in the corresponding layer operations. Each ***rule\_name*** specified in the rule file must be used only in one Layout Cell Match Rule statement. The entire list of matching rule names is concatenated into a single list for each matching layer operation, respectively.

The following material discusses how cells are determined to be *geometrically equal* in the context of this specification statement. For the default method of determining geometric equivalence, see the [Extent Cell](#), [Inside Cell](#), or [Not Inside Cell](#) discussions of the WITH MATCH keyword. The main difference between the default behavior and Layout Cell Match Rule is the latter determines equivalence based upon specified layers, while the default uses all layers for cell matching.

Consider the following:

**INSIDE CELL layer A<sub>1</sub> A<sub>2</sub> ... A<sub>m</sub> WITH MATCH [GOLDEN] RULE match\_rule**

**LAYOUT CELL MATCH RULE match\_rule BY LAYER L<sub>1</sub> [INSIDE OF LAYER W<sub>1</sub> ]**

...

**LAYER L<sub>P</sub> [INSIDE OF LAYER W<sub>P</sub> ]**

Equality of cells A (unplaced, and, if GOLDEN is specified in the layer operation, also in the “golden” cell database) and cells C (placed) is determined as follows. For every layer L in { L<sub>1</sub>, ..., L<sub>p</sub> } (and only in that set of layers):

1. Flatten and merge all geometry on L in cell A and its sub-hierarchy into the coordinate space of A.
2. Flatten and merge all geometry on L in cell C and its sub-hierarchy into the coordinate space of C.
3. If the resulting geometry set in either of the steps 1 or 2 is empty, then the cells are not equal.
4. Perform a Boolean XOR of the geometry in step 1 with the geometry in step 2. If the XOR result is non-empty, then the cells are not equal.

If INSIDE OF LAYER W is specified, then the primary algorithm is modified as follows:

1. Flatten and merge all geometry on L in cell A and its sub-hierarchy into the coordinate space of A.
2. Flatten and merge all geometry on L in cell C and its sub-hierarchy into the coordinate space of C.
3. Flatten and merge all geometry on layer W in cell A and its sub-hierarchy into the coordinate space of A and copy the resultant W geometry from A into the top-level space of cell C.
4. Perform a Boolean AND of the resultant geometry of step 1 in cell A with the resultant geometry of step 3 in A.

If the resulting geometry set in any of the steps 1 through 4 is empty, then the cells are not equal.

5. Perform a Boolean AND of the resultant geometry of step 2 in cell C with the resultant geometry of step 3 in C.
6. Perform a Boolean XOR of the geometry in step 4 with the geometry in step 5. If the XOR result is non-empty, then the cells are not equal.

## Examples

### Example 1

In this example, layout.gds is expected to have a cell that is geometrically equivalent to ramcell, which is a known cell name from golden.gds. The equivalent cell in layout.gds might not share the name ramcell. The Layout Cell Match Rule statement instructs the cell match to be made based upon the met1 layer.

```
LAYOUT PATH layout.gds // main design
LAYOUT PATH golden.gds GOLDEN // cell library for matching cell names

// use golden cell matching based upon the met1 layer only
LAYOUT CELL MATCH RULE met1_rule BY LAYER met1

// if the layer met1 in ramcell in golden.gds is equivalent to some
// cell in layout.gds, then derive x.
x = INSIDE CELL met1 ramcell WITH MATCH GOLDEN RULE met1_rule
```

## Layout Clone By Layer

Specification statement

**LAYOUT CLONE BY LAYER** *layer* [*layer...*]

### Parameters

- *layer*

A required original layer name or number that causes cells to be cloned based upon the presence of shapes on the layer in a cell. More than one layer may be specified.

### Description

Specifies that cells are to be cloned during the internal hierarchical database construction phase of hierarchical Calibre applications based upon interaction with shapes on the specified *layer* parameters. Cell cloning can be useful in avoiding flattening of the Calibre hierarchical database in certain types of flows that use marker layers.

This command is very useful if marker shapes are used to distinguish low power, general purpose, and high performance cells that are usually the same cells functionally, but contain different gate sizing and treatment of implant or other processing steps. Because cells covered by the marker layer are processed differently from cells not covered, both the covered and uncovered instances are flattened during processing.

This command allows the hierarchical database constructor to clone the cells based on layer markers, which reduces the flattening that may otherwise occur during the run. The marker layers are candidates for use as *layer* parameters in this statement.

Layout Clone By Layer may be specified any number of times.

---

#### Note



This statement should only be used if your design conforms to the criteria outlined otherwise performance degradation can occur.

---

This statement works as follows for each of the *layer* parameters, in the order specified. Let MKR be a layer parameter of a Layout Clone By Layer specification statement. Then the following occur:

1. All geometry on MKR is temporarily flattened into the top-level cell.
2. All placements that cut (that is, are not either inside of or outside of) flattened geometry on MKR are expanded recursively from the top level down. After this step, all cell placements from the flat (that is, top-level cell) point-of-view are either inside of or outside of MKR geometry.
3. Cells which, from the flat point-of-view, have a placement both inside of and outside of geometry on MKR are then cloned into two distinct cells: one cell has all placements inside of (flat view) MKR layer geometry, and the other cell has all placements outside of (flat view) MKR layer geometry. Each cloned cell name is the original cell name with a string of the form \$Lnumber\$ appended.

## Limitations

Because each selected placement from the algorithm described previously is cloned as a unique cell definition, results will be larger.

Result counts for cloned versus non-cloned runs will not be equivalent for any rule check that involved cloned data. In most cases, an XOR of geometric results shows that the results are accurate.

## Examples

The following statement applies cloning of cells containing the specified layers:

```
LAYOUT CLONE BY LAYER marker1 marker2
```

# Layout Clone Rotated Placements

Specification statement

## LAYOUT CLONE ROTATED PLACEMENTS {NO | YES}

### Parameters

- **NO**

Required keyword that specifies placements rotated by 90 or 270 degrees are not cloned. This is the default behavior.

- **YES**

Required keyword that applies the cloning process only when the transform swaps horizontal and vertical axes, that is, contains a 90-degree or 270-degree rotation.

### Description

Specifies whether to clone cells that are rotated by 90 or 270 degrees; no other transforms are considered.

---

**Note**

This statement should only be used if your design conforms to the criteria outlined in [Layout Clone Transformed Placements](#) otherwise performance degradation can occur.

---

Layout Clone Rotated Placements YES has a similar application as [Layout Clone Transformed Placements](#) YES for avoiding excessive hierarchical degradation in long layer derivation flows that distinguish edges based upon angular orientation. This statement is more limited than Layout Clone Transformed Placements YES because Layout Clone Rotated Placements YES applies the cloning process only when the transform is a 90- or 270-degree rotation. Thus, all placements end up with a 0- or 180-degree rotation, with or without an x-axis reflection. Every cell may have only one clone. Layout Clone Rotated Placements YES is generally more efficient than Layout Clone Transformed Placements YES in that it creates fewer cloned cells.

If Layout Clone Transformed Placements YES is specified, it overrides Layout Clone Rotated Placements YES. That is, if Layout Clone Transformed Placements YES is specified, then all non-identity transforms are removed regardless of whether Layout Clone Rotated Placements YES is specified.

If Layout Clone Transformed Placements NO is explicitly specified in the rule file (that is, the statement is not simply enabled by default), then it overrides Layout Clone Rotated Placements YES, and no cell cloning is performed.

Litho operations that are rotation sensitive such as DDL will trigger rotation cloning even if Layout Clone Rotated Placements NO is specified.

### Limitations

- Because each transformed placement is cloned as a unique cell definition, results output will be larger.

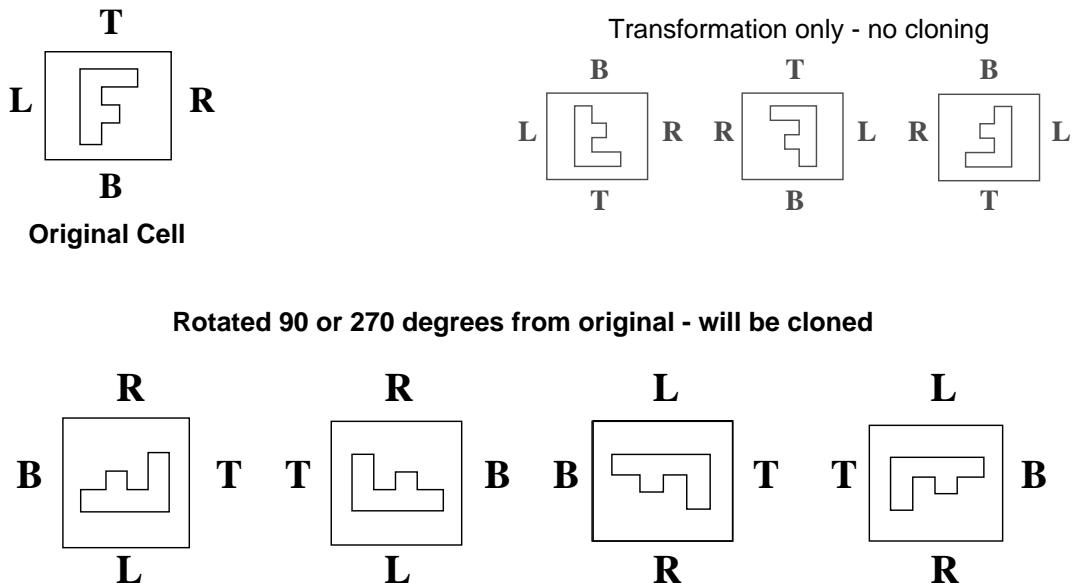
- Result counts for cloned versus non-cloned runs will not be equivalent for any rule check that involved cloned data. In most cases, an XOR of geometric results shows that the results are accurate.
- If the rule is generating geometric data, other general limitations for the results variance also apply. Therefore, if REGION CENTERLINE or REGION output is specified from dimensional check operations, the geometric results of this may change by 1 or 2 dbu.
- If the input data uses a large percentage of transformed placements, and the input size is relatively small, the Calibre application performance may not improve.

## Examples

The following statement applies the cloning process when the transform swaps horizontal and vertical axes, that is, contains a 90-degree or 270-degree rotation:

```
LAYOUT CLONE ROTATED PLACEMENTS YES
```

**Figure 4-142. Rotated Placements**



# Layout Clone Transformed Placements

Specification statement

## LAYOUT CLONE TRANSFORMED PLACEMENTS {NO | YES}

### Parameters

- **NO**

Required keyword that specifies cell cloning does not occur. This is the default behavior.

- **YES**

Required keyword that specifies all placements end up with identity transforms and every cell may have up to seven clones.

### Description

Specifies whether to use cell cloning during the hierarchical database (HDB) construction phase of a Calibre run.

In general, some combinations of SVRF layer operations and input data structures show marked performance improvement when Layout Clone Transform Placements YES is specified (the default is NO). Specifically, improved performance may be observed if all of the following are true:

- The input design contains rotated or reflected placements.
- The rule file contains use of the “==” constraint in the [Angle](#) layer operation, or the ANGLED secondary keyword is used in dimensional check operations. Directional operations such as [Grow](#), [Shrink](#), or [Shift](#) often appear in conjunction with the angle orientation derivations.

---

**Note**

This statement should only be used if your design conforms to the criteria outlined above otherwise performance degradation can occur.

---

Specifying YES is primarily useful in avoiding excessive hierarchical degradation (flattening) during long layer derivation flows having layer operations that distinguish edges by 0-degree versus 90-degree orientation. Assist feature generation is one example. Specifying YES is not generally recommended if your design and rule deck do not have the characteristics described previously.

During the HDB construction phase, Calibre applications must eliminate placements with floating-point transforms. These include non-orthogonal rotation or non-unit magnification. Placements with non-orthogonal rotation are eliminated by expansion.

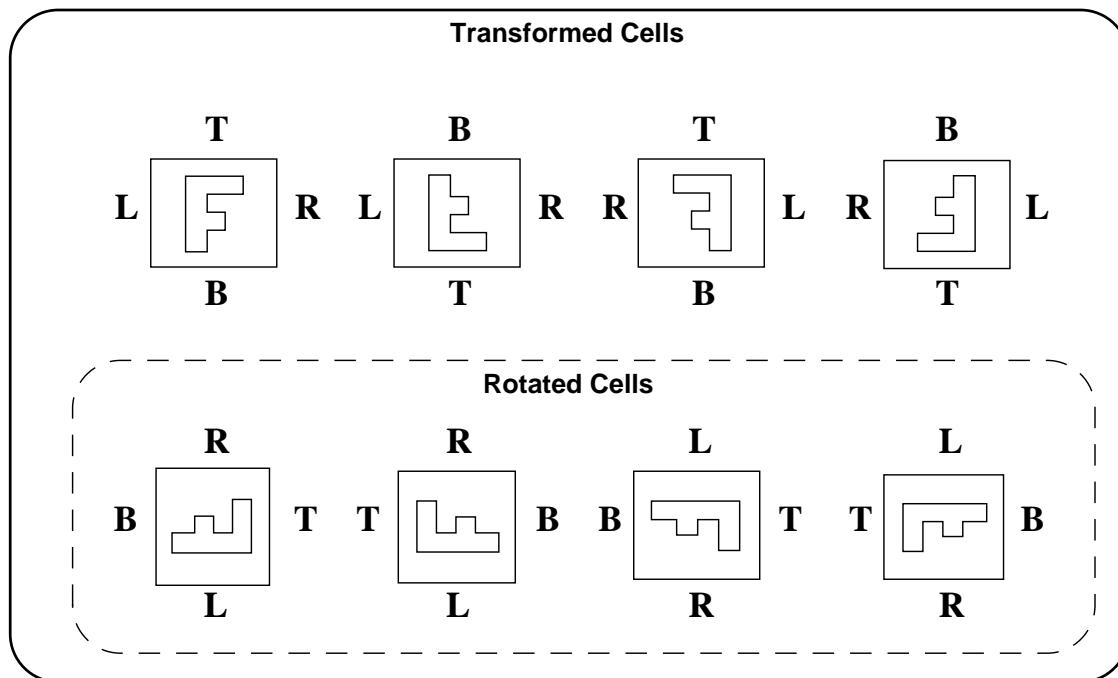
Elimination of non-unit magnification is more complicated and is done using a process called cloning. The magnification is removed by applying it to the contents of the placement's cell. Because the cell now differs from the original, a new copy (or clone) must be created, and the placement now becomes a placement of the clone. Magnification applied within the cloned cell

itself creates magnified placements, so the removal of such placements must be applied recursively. This process recognizes duplicate clones, and if a cell is cloned in exactly the same way as all its placements, then it reverts to the original cell. Otherwise, the cloned cell name is the original cell name with \$Cnumber\$ appended.

The Layout Clone Transformed Placements YES statement directs Calibre applications to eliminate all non-identity (an identity transformation has an angle = 0 and no reflection) integer placement transforms during the HDB construction phase. When YES is specified, then all placements have identity transforms and every cell may have up to seven clones created internally. The same naming convention is used as described in the preceding paragraph. Few layout databases contain only translation in their placement transforms. However, in some cases it is absolutely critical that REGION CENTERLINE output consistency from dimensional check operations be maintained between flat and hierarchical Calibre runs and between multiple versions of Calibre. In such cases specifying YES is desirable.

**Layout Clone Rotated Placements** YES performs a similar function but on a more limited scale, cloning only those cells shown as Rotated Cells in [Figure 4-143](#). If you specify Layout Clone Transformed Placements NO explicitly (that is, this is not simply accepted as the default, but appears in the rule file), then no cloning is performed, even if Layout Clone Rotated Placements YES is specified.

**Figure 4-143. Transformed Versus Rotated Placements**



Litho operations that are affected by transforms such as process window OPC will trigger rotation cloning even if Layout Clone Transformed Placements NO is specified.

This statement may be specified only once.

### Limitations

- Because each transformed placement is cloned as a unique cell definition, results output will be larger.
- Result counts for cloned versus non-cloned runs will not be equivalent for any rule check that involved cloned data. In most cases, an XOR of geometric results shows that the results are accurate.
- If the rule is generating geometrical data, other general limitations for the results variance also apply. Therefore, if REGION CENTERLINE or REGION output is specified from dimensional check operations, the geometric results of this may change by 1 or 2 dbu.
- If the input data uses a large percentage of transformed placements, and the input size is relatively small, the Calibre application performance may not improve.

### Examples

```
// remove placements having non-zero transforms
LAYOUT CLONE TRANSFORMED PLACEMENTS YES
```

# Layout Depth

Specification statement

## LAYOUT DEPTH {ALL | PRIMARY}

Used only in Calibre.

### Parameters

- **ALL**

Keyword that specifies that shapes are read from the top-level cell to the bottom of the hierarchy. This is the default behavior when you do not include this statement in the rule file.

- **PRIMARY**

Keyword that specifies that shapes are read from the top-level cell only.

### Description

Specifies the hierarchical depth for processing geometric databases in Calibre. This statement has no effect when [Layout System](#) is SPICE or CNET. Not used for ASCII and BINARY format layout systems. Can be specified once in your rule file.

### Examples

#### Example 1

The following example specifies that geometries are read from the top-level cell to the bottom of the hierarchy.

```
LAYOUT DEPTH ALL //read geometry from all levels
```

#### Example 2

This example specifies that geometries are read from the top-level cell only.

```
LAYOUT DEPTH PRIMARY //read geometry only from top level
x = NET metals5 VSS //find VSS metals5
```

## Layout Error On Input

Specification statement

### LAYOUT ERROR ON INPUT {YES | NO}

Used only in Calibre.

#### Parameters

- **YES**

Keyword indicating that layout input module error/warning exceptions become fatal errors. This is the default behavior when you do not include this statement in the rule file.

- **NO**

Keyword that indicates the GDSII or OASIS input module error/warning exceptions *do not* become fatal errors.

#### Description

---

**Note**



This statement is superseded by [Layout Input Exception Severity](#).

---

Specifies whether certain warnings reported when reading a GDSII or OASIS layout database are to become fatal errors. These include warnings for:

- Missing or duplicate cell references.
- Absolute angle or magnification parameters in placement transforms.
- Non-orientable polygons. See [Figure 4-113](#) on page 605 for an example.
- Paths that are too large to expand (more than 1024 points).
- Paths with absolute width parameters.
- Problems extending type 4 paths. (A type 4 path is one with variable square ends.)
- Rule file precision does not match that specified in the database.

If Layout Error On Input NO is specified, the above exceptions produce warnings and processing continues. If Layout Error On Input YES is specified, these exceptions produce fatal errors. This statement may be specified once in your rule file.

#### Examples

##### Example 1

Basic specification.

```
LAYOUT ERROR ON INPUT YES
```

**Example 2**

This example shows a method for merging GDSII databases. The rule file excerpt merges multiple GDSII databases with top cells having the same name.

The files two.gds, three.gds, and four.gds are flat with the top cell named TOP. The result is a flat cell named TOP. The four layers being merged from the respective files are poly, diff, metal, and contact.

```
LAYOUT PATH two.gds three.gds four.gds
LAYOUT SYSTEM GDSII
LAYOUT PRIMARY "TOP"
LAYOUT ALLOW DUPLICATE CELL YES
LAYOUT ERROR ON INPUT NO

DRC RESULTS DATABASE "out.gds" GDSII
DRC SUMMARY REPORT "merge.rpt"

LAYER poly 1
LAYER diff 2
LAYER metal 3
LAYER contact 4

layer_1 {copy poly}
layer_2 {copy diff}
layer_3 {copy metal}
layer_4 {copy contact}

DRC CHECK MAP layer_1 GDSII 1
DRC CHECK MAP layer_2 GDSII 2
DRC CHECK MAP layer_3 GDSII 3
DRC CHECK MAP layer_4 GDSII 4
```

## Layout Ignore Text

Specification statement

**LAYOUT IGNORE TEXT {CELL LIST *name*}**  
[DATABASE RULES | DATABASE | RULES]  
[BY LAYER *layer*]

Used only in hierarchical Calibre.

### Parameters

- **CELL LIST *name***

A required cell list name specified with [Layout Cell List](#).

- **DATABASE RULES | DATABASE | RULES**

Optional keywords that control which sources of layout text are ignored.

DATABASE RULES — Specifies that text objects in the GDSII or OASIS layout database and those specified in the rule file using [Layout Text](#) specification statements are ignored. This is the default mode.

DATABASE — Specifies that text objects in the GDSII or OASIS database only are ignored.

RULES — Specifies that Layout Text statements specified in the rule file only are ignored.

- **BY LAYER *layer***

An optional keyword set that specifies text objects are ignored only on the specified *layer*. The *layer* parameter may be an original layer number, layer name, or layer set. If it is a layer set, then the semantics are equivalent to the statement being repeated for each layer in the set.

### Description

Ignores layout text for cells specified with *name*. The *name* parameter is the name of a list of cells defined with Layout Cell List. This statement operates on text in GDSII and OASIS databases, and text created using [Layout Text](#) specification statements by default. Connectivity extraction text objects and text objects created for [Not] [With Text](#) operations are affected by this statement. [Text](#) statement objects are not affected.

You control which sources of layout text are ignored by using the DATABASE and RULES keywords. The default is DATABASE RULES, which means both text objects in the layout database and Layout Text statements in the rule file are ignored.

Note that the BY LAYER *layer* parameter does not have to be declared in a [Text Layer](#) statement, a [Port Layer Text](#) statement, as a [Device](#) TEXT PROPERTY LAYER parameter, or any other kind of text layer. However, if a *layer* parameter is not used as a text layer of some kind, then all text on this layer is ignored.

The keywords of this statement act as filters. For example, if DATABASE and BY LAYER are both specified, then only the text that is in the specified cell list, and that originates from the input database, and that is on the specified *layer* is ignored.

Layout Ignore Text and [Layout Rename Text](#) statements (including those with multiple editing commands within the same statement) are applied to each text in the order these statements (and editing commands in the case of Layout Rename Text) are written in the rule file. As soon as one of the statements performs its specified action, such as edits or ignores text, subsequent statements are not applied to this text.

This statement has no effect in ICverify applications.

## Examples

### Example 1

```
// Ignore layout database text for cell names that start with
// the letter "a"
LAYOUT CELL LIST list1 "a*"
LAYOUT IGNORE TEXT CELL LIST list1
```

### Example 2

The following statements ignore all text on layer TXT1 in cells whose name starts with “B”:

```
LAYER TXT1 1
TEXT LAYER txt1
LAYOUT CELL LIST a "B*"
LAYOUT IGNORE TEXT CELL LIST a BY LAYER txt1
```

### Example 3

The following example demonstrates how the rules for multiple Layout Ignore Text and Layout Rename Text statements apply in presence of BY LAYER options.

```
LAYER txt1 1
LAYER txt2 2
LAYER txt3 3
TEXT LAYER txt1 txt2 txt3

LAYOUT CELL LIST a "A*"
LAYOUT CELL LIST all "*"

LAYOUT RENAME TEXT "/^X/X/" CELL LIST a BY LAYER txt1 TO txt2 // 1
LAYOUT IGNORE TEXT CELL LIST a BY LAYER txt1 // 2
LAYOUT RENAME TEXT CELL LIST a BY LAYER txt2 TO txt3 // 3
LAYOUT IGNORE TEXT CELL LIST all // 4
```

<b>Text</b>	<b>Layer</b>	<b>Cell</b>	<b>Result Text</b>	<b>Result Layer</b>	<b>Comment</b>
X1	txt1	A_CELL	X1	txt2	statement 1 applies
Y1	txt1	A_CELL	ignored	ignored	statement 2 applies

Text	Layer	Cell	Result Text	Result Layer	Comment
X1	txt2	A_CELL	X1	txt3	statement 3 applies (text from row 1 is not remapped)
Y1	txt2	A_CELL	Y1	txt3	statement 3 applies
X1	txt1	B_CELL	ignored	ignored	statement 4 applies

**Example 4**

The following statements cause the tool to ignore all text from the input database and to use only text from Layout Text statements:

```
LAYOUT CELL LIST all "*"
LAYOUT IGNORE TEXT CELL LIST all DATABASE // use only rule file text
```

**Example 5**

The following example shows how to read text on one layer at all levels of the hierarchy, while reading text from a second layer only at the top level.

```
LAYER txt1 1
LAYER txt2 2
TEXT LAYER txt1 txt2
TEXT DEPTH ALL // read text at all levels
                // filter txt 1 with LAYOUT IGNORE TEXT

// ignore txt1 at all levels except primary
LAYOUT CELL LIST all_but_top "*" "!TOP"
LAYOUT IGNORE TEXT CELL LIST "all_but_top" BY LAYER txt1
```

# Layout Input Exception RDB

Specification statement

## LAYOUT INPUT EXCEPTION RDB *filename*

[BY CELL | BY LAYER | BY EXCEPTION]

[TOP SPACE | CELL SPACE] [EXIT NONEMPTY] [MAG *value*]

Used only in Calibre.

## Summary

Specifies an ASCII results database (RDB) that contains selected input exceptions from [Layout Input Exception Severity ... RDB](#) statement settings in the rule file.

## Parameters

- ***filename***

Required pathname of the results database.

- **BY CELL**

Optional keyword specifying that each DRC result lists all the exceptions for each cell. Cells with no exceptions are excluded. This is the default. Each DRC result name is the name of the layout input database cell that contains the exceptions.

- **BY LAYER**

Optional keyword specifying that each DRC result lists all exceptions for each primitive layer number. Layers with no exceptions are excluded. Each DRC result name is the character L followed by the layer number.

- **BY EXCEPTION**

Optional keyword specifying that each DRC result lists all exceptions for each exception type. Exception types with no elements are excluded. Each DRC result name is the exception name, and is the same as that specified in the [Layout Input Exception Severity](#) statement.

- **TOP SPACE**

Optional keyword specifying that coordinate data is given in top-level coordinate space. This is the default. If TOP SPACE is specified, then the cell-to-world transform for each cell is that of one arbitrarily-selected placement of the cell in the flat viewpoint. This transform is consistent across the entire results database.

- **CELL SPACE**

Optional keyword specifying that coordinate data is given in cell coordinate space.

- **EXIT NONEMPTY**

Optional parameter specifying that the process is to abort immediately after the input layout database is read if any exceptions have been written to a Layout Input Exception Severity results database. An error message is generated if the program exits.

- **MAG value**

Optional parameter specifying that all coordinates output to the results database are magnified by the given *value*, where *value* is a positive undimensioned floating-point numeric (and may be a numeric expression containing variables).

### Description

Specifies an ASCII results database (RDB) that contains selected input exceptions from [Layout Input Exception Severity ... RDB](#) statement settings in the rule file.

During the stream-in phase of Calibre applications, certain input exceptions are flagged. These include such things as re-entrant polygons, degenerate rectangles, missing cell references, and other problems. The response to these exceptions ranges from quietly ignoring questionable objects, to warning and perhaps processing them in some fashion, or issuing a fatal error. You have control over the handling of many of these exceptions using the Layout Input Exception Severity statement.

The Layout Input Exception RDB statement allows you to visualize the selected exception objects. The geometric format of the objects varies by exception type. This RDB can be loaded into DRC RVE, just as any other RDB.

### Database Format

Each DRC result, regardless of RDB format, has three properties:

- CN property — the cell name of the object.
- E property — the exception name of the object.
- L property — the primitive layer number of the object for GDS and OASIS. If the object's datatype is non-zero then the L property value is of the form <layer>.<datatype> (similar to WORKbench format).

Coordinate data is either in cell coordinate space or top-level coordinate space depending on the parameters CELL SPACE and TOP SPACE (TOP SPACE is the default).

The RDB name is checked for collisions with [DRC Results Database](#) or [DRC Check Map](#) database names. A conflict generates a compiler error. The names of RDBs generated by other layer operations are not checked.

The RDB may have one of three canonical forms depending upon the presence of BY CELL, BY LAYER, or BY EXCEPTION. The default is BY CELL.

#### BY CELL Format

In the RDB, each DRC rule check name is the name of the layout input database cell that contains the exceptions.

```
TOP 1000
CHANSEQ    -- rule check name is a cell name
20 20 0 Dec 27 16:45:38 2007
p 1 6
CN CHANSEQ c
L 34
```

```

E POLYGON_NONSIMPLE
87600 92400
76500 87600
...
p 2 9
CN CHANSEQ c
L 3.2
E PATH_ENDSEGMENT_SHORT
5670 7890
2670 4280
...
MUX_CELL <-- rule check name is a cell name
17 17 0 Dec 27 16:45:38 2007
...

```

**BY LAYER Format**

In the RDB, each DRC rule check name is the character L followed by the layer number.

```

TOP 1000
L34 <-- rule check name is a layer number
20 20 0 Dec 27 16:45:38 2007
p 1 6
CN CHANSEQ c
L 34
E POLYGON_NONSIMPLE
87600 92400
76500 87600
...
p 2 15
CN RAMBIT4 c
L 34.17
E PATH_CIRCULAR
3450 18900
6780 18920
...
L63 <-- rule check name is a layer number
2 2 0 Dec 27 16:45:38 2007
...

```

**BY EXCEPTION Format**

In the RDB, each DRC rule check name is the exception name and is the same as that specified in the Layout Input Exception Severity ... RDB statement.

```

TOP 1000
POLYGON_NONSIMPLE <-- rule check name is an exception name
98 98 0 Dec 27 16:45:38 2007
p 1 6
CN CHANSEQ c
L 34.5
E POLYGON_NONSIMPLE
87600 92400
76500 87600
...
p 2 17
CN DATACOL c
L 22
E POLYGON_NONSIMPLE

```

## Layout Input Exception RDB

---

```
4560 6780
1800 1900
...
PATH_ACUTE    <-- rule check name is an exception name
18 18 0 Dec 27 16:45:38 2007
```

The name of the Layout Input Exception RDB file, if present, is placed as an empty DRC rule check at the end of the global [DRC Results Database](#), if ASCII-type. This is intended as a reminder that an RDB has been created during the run. If the Layout Input Exception RDB file actually contains DRC results, then the extent of the first result is added as a geometric result, along with an extra line of check text, to the rule check at the end of the global ASCII DRC results database. For example:

```
... -4387000 -4794250 -4387000 -4791750
-4385750 -4796977 -4385750 -4794250
LAYOUT_INPUT_EXCEPTION_RDBS
1 1 2 Jul 29 10:51:09 2008 input_exception.rdb
NOTE: The LAYOUT INPUT EXCEPTION RDB is non-empty <-- check text
p 1 4                                         <-- first RDB result
-815004 -3714005
-796995 -3714005
-796995 -3703995
-815004 -3703995
```

## Examples

```
LAYOUT INPUT EXCEPTION SEVERITY PATH_ACUTE 3 RDB
LAYOUT INPUT EXCEPTION RDB exceptions.rdb // store exception results
```

# Layout Input Exception Severity

Specification statement

**LAYOUT INPUT EXCEPTION SEVERITY** *exception\_name severity\_value* [RDB]

Used only in Calibre.

## Parameters

- *exception\_name*  
Case-insensitive member of a well-defined set of exceptions enumerated below.
- *severity\_value*  
Non-negative integer that allows you to individually control the severity of selected exceptions. Severity may include complete quiet, cautionary notes, warnings, or (fatal) errors.
- RDB  
Optional keyword that specifies any objects encountered as exceptions are written to the results database specified in the [Layout Input Exception RDB](#) statement.

## Description

Specifies an exception type and how the exception impacts the reading of a layout database through a user-selected *severity\_value*. This statement may be specified once per *exception\_name*. It is a compilation error for the same *exception\_name* to appear in multiple Layout Input Exception Severity statements. It is also a compilation error for a *severity\_value* in a Layout Input Exception Severity statement to be outside the range of severities for the particular exception.

---

### Note



Use this statement with care, particularly if you use an unfamiliar rule file that has different exception settings than the defaults.

---

This statement supersedes [Layout Error On Input](#).

The RDB keyword identifies exceptions to output to the results database specified in the Layout Exception Severity RDB statement. The RDB keyword is only permitted with geometric-type exceptions and certain *severity\_value* parameters (depending on exception type). The RDB keyword is not allowed with a fatal error severity setting. It is a compilation error for any Layout Input Exception Severity specification statement to have a RDB parameter if there is no global Layout Input Exception RDB specified.

[Table 4-15](#) enumerates the exceptions detected during reading of the input layout database(s) by Calibre applications. They are divided into categories:

- Exception Name — the *exception\_name* as it should appear in the Layout Input Exception Severity specification statement.
- [Layout System](#) — the layout system in which the exception may occur.

- Description — an explanation of the exception.
- Severities — a list of *severity\_value* settings and the default *severity\_value* (defaults always underlined).
- Backward — discusses how the exception interacts with the Layout Error On Input statement and other related specification statements.
- RDB support — these comments discuss RDB output for exceptions that support this.

**Table 4-15. Exceptions**

Arrays
<p>Exception Name: <b>ARRAY_PITCH_ZERO</b>            Layout System: GDSII and OASIS            Description: Controls zero row or column spacing in placement arrays in GDSII- or OASIS-type input layout databases. For GDSII, this occurs when an AREF placement has its x-dimension (or y-dimension) &gt; 1 but the spacing in the x-direction (or y-direction) is zero. For OASIS, this occurs when a placement record has a type 1 or 2 repetition and the x-direction spacing is zero, or a placement record has a type 1 or 3 repetition and the y-direction spacing is zero.            Severities:            0: Quietly process the placement as normal. (default)            1: Warn and process the placement as normal.            2: Error.            Backward: No interaction with Layout Error On Input.            RDB support for severities 0 and 1.            RDB output is a <math>4 \times 4</math> dbu square having a center at the array base point.</p>
AREF
<p>Exception Name: <b>AREF_ANGLLED</b>            Layout System: GDSII            Description: Controls handling of AREF rotation that is not a multiple of 90 degrees.            Severities:            0: Quietly process the AREF placement as normal. (default)            1: Warn and process the AREF placement as normal.            2: Error.            Backward: No interaction with Layout Error On Input.</p>
AREF_PLACEMENT
<p>Exception Name: <b>AREF_PLACEMENT</b>            Layout System: GDSII            Description: Controls handling of AREF structures.            Severities:            0: Quietly process the AREF placement as normal. (default)            1: Warn and process the AREF placement as normal.            2: Error.            Backward: No interaction with Layout Error On Input.</p>

**Table 4-15. Exceptions (cont.)**

Cells
<p>Exception Name: <b>DUPPLICATE_CELL</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A cell structure has been encountered multiple times in the input layout database.            Severities:            0: Ignore subsequent definitions.            1: Warn and ignore subsequent definitions.  <u>2</u>: Error. (default)            3: Quietly combine multiple definitions of the same cell.            4: Warn and combine multiple definitions of the same cell.            Backward: No interaction with Layout Error on Input.  <a href="#">Layout Allow Duplicate Cell</a> YES is equivalent to Layout Input Exception Severity DUPPLICATE_CELL 3 if the DUPLICATE_CELL exception is not otherwise set.            See also <a href="#">Layout Rename ICV</a>.</p>
<p>Exception Name: <b>EXCLUDE_CELL_NAME</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: An <a href="#">Exclude Cell</a> parameter is not located in the input layout database.            Severities:            0: Ignore the condition.  <u>1</u>: Warn for each non-located name. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>
<p>Exception Name: <b>EXTENT_CELL</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: An <a href="#">Extent Cell</a> parameter is not located in the input layout database.            Severities:            0: Ignore the condition.  <u>1</u>: Warn for each non-located name. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>
<p>Exception Name: <b>INSIDE_CELL</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A [NOT] <a href="#">Inside Cell</a> parameter is not located in the input layout database.            Severities:            0: Ignore the condition.  <u>1</u>: Warn for such non-located name. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>MISSING_REFERENCE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A placement contains no corresponding cell definition.            Severities:            0: Quietly generate an empty cell definition for the condition.            1: Warn and generate an empty cell definition for the condition.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
<p>Exception Name: <b>TOP_CELL_CHOSEN</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: The primary cell was selected by wildcard matching instead of exact comparison with the <a href="#">Layout Primary</a>[2] parameter. If options 1 or 2 are specified, the alternate choices for top cells are listed in NOTE messages in the transcript.            Severities:            0: Ignore the condition.  <u>1</u>: Warn if this happens. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>
<p><b>File I/O</b></p> <p>Exception Name: <b>LAYER_DIRECTORY</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A directory specified in a <a href="#">Layer Directory</a> statement cannot be opened, or a hierarchical Calibre application is not being run in 64-bit mode when using a Layer Directory statement.            Severities:            0: Quietly use memory-based layers.            1: Warn and use memory-based layers.  <u>2</u>: Error. (default)            Backward: No interaction with Layout Error On Input.</p>
<p><b>Layers</b></p> <p>Exception Name: <b>DATATYPE_MAP_SOURCE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: Geometric objects unmapped by any <a href="#">Layer Map</a> statement exist on required simple layer N but layer N is itself a source layer of a Layer Map statement.            Severities:            0: Ignore the condition.  <u>1</u>: Warn for each such target layer N. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>DATATYPE_MAP_TARGET</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: Geometric objects unmapped by any <a href="#">Layer Map</a> statement exist on required simple layer N but layer N is itself the target layer of a Layer Map statement.            Severities:            0: Ignore the condition.  <u>1</u>: Warn for each such target layer N. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>
<p>Exception Name: <b>EMPTY_LAYER</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: Simple layers required by the Calibre flow are empty. A simple layer is any layer number after application of pertinent datatype maps.            Severities:            0: Ignore the condition.  <u>1</u>: List the empty simple layers in the transcript after all of the input layout database files are read. (default)            2: Warn for each empty simple layer after all of the input layout database files are read.            3: Warn globally after all of the input layout database files are read.            4: Error globally after all of the input layout database files are read.            5: Combine severities 1 and 3.            6: Combine severities 1 and 4.            Backward: No interaction with Layout Error On Input.</p>
<p>Exception Name: <b>LAYER_OVER_BUMP</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: Used in dual-database comparison applications. Controls handling of objects from the first input layout database whose primitive layer number is greater than or equal to the <a href="#">Layout Bump2</a> specification statement value. These objects are not processed.            Severities:            0: Ignore the condition.  <u>1</u>: Transcript a list of all layer numbers in the first input layout database over the bump value. (default)            2: Warn for each layer number in the first input layout database over the bump value.            3: Warn globally that this condition exists.            4: Error globally if this condition exists.            5: Combine severities 1 and 3.            6: Combine severities 1 and 4.            Backward: No interaction with Layout Error On Input.</p>

**Table 4-15. Exceptions (cont.)**

Exception Name: **UNUSED\_DATA**

Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS

Description: A (layer,datatype) coordinate contains geometric objects that are not required by the flow.

Severities:

- 0: Ignore the condition.
- 1: Transcript a list of these coordinates after all of the input layout database files are read. (default)
- 2: Warn for each (layer,datatype) coordinate after all of the input layout database files are read.
- 3: Warn globally after all of the input layout database files are read.
- 4: Error globally after all of the input layout database files are read.
- 5: Combine severities 1 and 3.
- 6: Combine severities 1 and 4.

Backward: No interaction with Layout Error On Input.

#### Miscellaneous geometry

Exception Name: **CIRCLE\_RADIUS\_ZERO**

Layout System: OASIS

Description: An OASIS circle has zero radius.

Severities:

- 0: Ignore the circle.
- 1: Warn and ignore the circle. (default)
- 2: Error.

Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.

RDB support for severities 0 and 1.

RDB output is a  $4 \times 4$  dbu square having a center point on the object.

Exception Name: **CIRCLE**

Layout System: OASIS

Description: An OASIS circle has been encountered.

Severities:

- 0: Ignore the circle.
- 1: Warn and ignore the circle.
- 2: Error. (default)
- 3: Quietly replace the circle with a 64-vertex polygon.
- 4: Warn and replace the circle with a 64-vertex polygon.

Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.

**Table 4-15. Exceptions (cont.)**

<p><b>Exception Name:</b> <b>CTRAPEZOID_AREA_ZERO</b></p> <p><b>Layout System:</b> OASIS</p> <p><b>Description:</b> A CTRAPEZOID has zero area.</p> <p><b>Severities:</b></p> <ul style="list-style-type: none"> <li>0: Ignore the ctrapezoid.</li> <li><u>1</u>: Warn and ignore the ctrapezoid. (default)</li> <li>2: Error.</li> </ul> <p><b>Backward:</b> No interaction with Layout Error On Input.</p> <p><b>RDB support</b> for severities 0 and 1.</p> <p><b>RDB output</b> is a <math>4 \times 4</math> dbu square having a center point on the object.</p>
<p><b>Exception Name:</b> <b>CTRAPEZOID_DEGENERATE</b></p> <p><b>Layout System:</b> OASIS</p> <p><b>Description:</b> A CTRAPEZOID record is degenerate (that is, cannot be formed into a polygon as specified).</p> <p><b>Severities:</b></p> <ul style="list-style-type: none"> <li>0: Ignore the ctrapezoid.</li> <li>1: Warn and ignore the ctrapezoid.</li> <li><u>2</u>: Error. (default)</li> </ul> <p><b>Backward:</b> No interaction with Layout Error On Input.</p> <p><b>RDB support</b> for severities 0 and 1.</p> <p><b>RDB output</b> is a <math>4 \times 4</math> dbu square having a center point on the object.</p>
<p><b>Exception Name:</b> <b>TRAPEZOID_AREA_ZERO</b></p> <p><b>Layout System:</b> OASIS</p> <p><b>Description:</b> A TRAPEZOID has zero area.</p> <p><b>Severities:</b></p> <ul style="list-style-type: none"> <li>0: Ignore the trapezoid.</li> <li><u>1</u>: Warn and ignore the trapezoid. (default)</li> <li>2: Error.</li> </ul> <p><b>Backward:</b> No interaction with Layout Error On Input.</p> <p><b>RDB support</b> for severities 0 and 1.</p> <p><b>RDB output</b> is a <math>4 \times 4</math> dbu square having a center point on the object.</p>
<p><b>Exception Name:</b> <b>TRAPEZOID_DEGENERATE</b></p> <p><b>Layout System:</b> OASIS</p> <p><b>Description:</b> A TRAPEZOID record is degenerate (that is, cannot be formed into a polygon as specified).</p> <p><b>Severities:</b></p> <ul style="list-style-type: none"> <li>0: Ignore the trapezoid.</li> <li>1: Warn and ignore the trapezoid.</li> <li><u>2</u>: Error. (default)</li> </ul> <p><b>Backward:</b> No interaction with Layout Error On Input.</p> <p><b>RDB support</b> for severities 0 and 1.</p> <p><b>RDB output</b> is a <math>4 \times 4</math> dbu square having a center point on the object.</p>

**Table 4-15. Exceptions (cont.)**

<p><b>Exception Name: XGEOMETRY_UNSUPPORTED</b>  <b>Layout System:</b> OASIS  <b>Description:</b> An XGEOMETRY record (not supported) has been encountered.  <b>Severities:</b></p> <ul style="list-style-type: none"> <li>0: Ignore the xgeometry.</li> <li>1: Warn and ignore the xgeometry.</li> <li><u>2</u>: Error. (default)</li> </ul> <p>Backward: No interaction with Layout Error On Input.</p>
<p><b>Paths</b></p> <p><b>Exception Name: PATH_ACUTE</b>  <b>Layout System:</b> GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS  <b>Description:</b> A path has a centerline with one or more acute angles.  <b>Severities:</b></p> <ul style="list-style-type: none"> <li>0: Quietly ignore the path.</li> <li><u>1</u>: Quietly process the path as usual. (default)</li> <li>2: Warn and ignore the path.</li> <li>3: Warn and process the path as usual.</li> <li>4: Fatal error.</li> <li>5: Quietly process the path, clipping 45-degree acute angles on the centerline so as to avoid small angled notches (as shown in <a href="#">Figure 4-144</a>) which may result from the default expansion algorithm.</li> <li>6: Warn and process the path, clipping 45-degree acute angles on the centerline so as to avoid small angled notches (as shown in <a href="#">Figure 4-144</a>) which may result from the default expansion algorithm.</li> </ul> <p>Backward: No interaction with Layout Error On Input.  RDB supported for severities 0, 1, 2, and 3.  RDB output is merged expansion by 4 dbu of the centerline segments forming one acute angle.</p>
<p><b>Exception Name: PATH_ANGLLED</b>  <b>Layout System:</b> GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS  <b>Description:</b> A path has a centerline with one or more angled segments.  <b>Severities:</b></p> <ul style="list-style-type: none"> <li>0: Quietly ignore the path.</li> <li><u>1</u>: Quietly process the path as usual. (default)</li> <li>2: Warn and ignore the path.</li> <li>3: Warn and process the path as usual.</li> <li>4: Fatal error.</li> </ul> <p>Backward: No interaction with Layout Error On Input.  RDB supported for severities 0, 1, 2, and 3.  RDB output is expansion by 4 dbu of one angled centerline segment.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>PATH_BIG</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A path has more than 1024 vertices.            Severities:            0: Ignore the path.            1: Warn and ignore the path.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
<p>Exception Name: <b>PATH_CIRCULAR</b>            Layout System: GDSII, LEF/DEF, MILKYWAY, OPENACCESS            Description: A type-1 path (circular endcap) is present.            Severities:            0: Quietly drop the path.  <u>1</u>: Quietly convert the path to type 0 (flush ended). (default)            2: Quietly convert the path to type 2 (square ended).            3: Drop the path with a warning.            4: Warn and convert the path to type 0.            5: Warn and convert the path to type 2.            6: Error.            7: Quietly create a round-ended path.            8: Warn and create a round-ended path.            Backward: No interaction with Layout Error On Input.            RDB support for severities 0, 1, 2, 3, 4, 5, 7 and 8.            RDB output is the merged expansion of the centerline segments by 4dbu.            Note: Round-ended paths are created by merging together the path expanded as if it were type 0 and two 64-vertex polygons centered on each endpoint with a radius of half the path width.</p>
<p>Exception Name: <b>PATH_COINCIDENT</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A path has a centerline with coincident endpoint coordinates.            Severities:            0: Quietly ignore the path.  <u>1</u>: Quietly process the path as usual. (default)            2: Warn and ignore the path.            3: Warn and process the path as usual.            4: Fatal error.            Backward: No interaction with Layout Error On Input.            RDB support for severities 0, 1, 2, and 3.            RDB output is merged expansion by 4 dbu of the first and last centerline segments.</p>

**Table 4-15. Exceptions (cont.)**

Exception Name: **PATH\_DEGENERATE**

Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS

Description: A path has 0 or 1 vertices.

Severities:

- 0: Ignore the path.
- 1: Warn and ignore the path.
- 2: Error. (default)

Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.

RDB supported for severities 0 and 1.

Zero-vertex objects are not output. Output for a single vertex object consists of a 4-dbu box around the vertex.

Exception Name: **PATH\_ENDSEGMENT\_SHORT**

Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS

Description: The endsegment of a type-0 GDSII path or flush-ended OASIS path is less than 1/2 the path width.

Severities:

- 0: Quietly expand the path, even if a small notch develops.
- 1: Warn and expand the path, even if a small notch develops. (default)
- 2: Error
- 3: Quietly extend the short endsegment to avoid a notch prior to expanding the path (see [Figure 4-145](#)).
- 4: Warn and extend the short endsegment to avoid a notch prior to expanding the path.

Backward: No interaction with Layout Error On Input.

RDB support for severities 0, 1, 3 and 4.

RDB output is expansion of each short endsegment by 4 dbu.

Exception Name: **PATH\_EXTENSION**

Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS

Description: A type-4 GDSII path or variable-end OASIS path cannot be extended.

Severities:

- 0: Quietly do not perform variable-end path extension.
- 1: Warn and do not perform variable-end path extension.
- 2: Error. (default)

Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.

RDB support for severities 0 and 1.

RDB output is expansion of each problematic endsegment by 4 dbu.

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>PATH_NONORIENTABLE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A path cannot be oriented after expansion.            Severities:            0: Ignore the path.            1: Warn and ignore the path.  <u>2</u>: Error. (default)            3: Fix the path using a double-merge technique. This technique generally succeeds in filling non-orientable polygons in an intuitive fashion. The polygon is processed with no message.            4: Fix the path using a double-merge technique. This technique generally succeeds in filling non-orientable polygons in an intuitive fashion. The polygon is processed with a warning message.            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.            RDB support for severities 0, 1, 3, and 4.            RDB output is merged expansion of the centerline segments by 4 dbu.</p>
<p>Exception Name: <b>PATH_NONSIMPLE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A path overlaps itself after expansion.            Severities:  <u>0</u>: Ignore the condition since the overlap is merged away. (default)            1: Warn and ignore the condition since the overlap is merged away.            2: Error.            Backward: <a href="#">Flag Nonsimple Path</a> YES sets severity 1. No interaction with Layout Error On Input.            RDB support for severities 0 and 1.            RDB output is merged expansion of the centerline segments by 4 dbu.</p>
<p>Exception Name: <b>PATH_SPIKE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A path retraces itself for a portion of the length.            Severities:            0: Quietly remove the spike.            1: Warn and remove the spike.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.            RDB support for severities 0 and 1.            RDB output is a Boolean AND expansion of the centerline segments forming the spike.</p>

**Table 4-15. Exceptions (cont.)**

<p><b>Exception Name: PATH_VARIABLE</b></p> <p>Layout System: GDSII, OASIS</p> <p>Description: A GDSII path of type 4 or OASIS path with a non-flush or non-square extension is encountered. The location of the path, containing cell name, and layer and datatype numbers are placed into the message. Output to the Layout Input Exception RDB file is the merged expansion of the centerline segments by 4 dbu.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li>0: Quietly drop the path.</li> <li>1: Quietly process the path as normal (default).</li> <li>2: Warn and drop the path.</li> <li>3: Warn and process the path as normal.</li> <li>4: Fatal error.</li> </ul> <p>Backward: No interaction with Layout Error On Input. RDB support for severities 0, 1, 2, and 3.</p>
<p><b>Exception Name: PATH_WIDTH_ABSOLUTE</b></p> <p>Layout System: GDSII</p> <p>Description: A GDSII path has absolute width.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li>0: Ignore the path.</li> <li>1: Warn and ignore the path.</li> <li>2: Error. (default)</li> </ul> <p>Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
<p><b>Exception Name: PATH_WIDTH_ODD</b></p> <p>Layout System: GDSII</p> <p>Description: A path has an odd-valued width. The half-width value is rounded when the path is expanded to a polygon.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li>0: Quietly process the path. (default)</li> <li>1: Warn and process the path.</li> <li>2: Error.</li> </ul> <p>Backward: No interaction with Layout Error On Input. RDB supported for severities 0 and 1. RDB output is merged expansion of the centerline segments by 4 dbu.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>PATH_WIDTH_ZERO</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A path has zero width.            Severities:            0: Ignore the path.  <u>1</u>: Warn and ignore the path. (default)            2: Error.            Backward: No interaction with Layout Error On Input.            RDB supported for severities 0 and 1.            RDB output is merged expansion of the centerline segments by 4 dbu.</p>
<b>Placements</b>
<p>Exception Name: <b>PLACEMENT_ANGLE_ABSOLUTE</b>            Layout System: GDSII            Description: A GDSII placement has an absolute rotation angle.            Severities:            0: Ignore the rotation.            1: Warn and ignore the rotation.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
<p>Exception Name: <b>PLACEMENT_MAGNIFICATION_ABSOLUTE</b>            Layout System: GDSII            Description: A GDSII placement has an absolute magnification.            Severities:            0: Ignore the magnification.            1: Warn and ignore the magnification.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
<p>Exception Name: <b>PLACEMENT_MAGNIFICATION_NONPOSITIVE</b>            Layout System: GDSII            Description: A placement has a non-positive magnification.            Severities:            0: Ignore the magnification.            1: Warn and ignore the magnification.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>PLACEMENT_MAGNIFIED</b>            Layout System: GDSII or OASIS            Description: A GDSII or OASIS placement has a non-unit magnification. The value of the magnification, cell name of the placement, location of the placement, and the containing cell name are placed into the message. For OASIS the message includes the file name and record offset but omits the placement cell name.            Severities:            0: Quietly process the magnified placement as normal. (default)            1: Warn and process the magnified placement as normal.            2: Error.            Backward: None.</p>
<p><b>Polygons</b></p> <p>Exception Name: <b>LAYOUT_POLYGON</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A cell name is specified in a <a href="#">Layout Polygon</a> statement, but the cell name is not in the layout database.            Severities:            0: Ignore the polygon.            1: Warn and ignore polygon. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>
<p>Exception Name: <b>POLYGON_ACUTE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: Acute boundary exists in a GDSII-type database or an acute polygon exists in an OASIS database.            Severities:            0: Ignore the object. (default)            1: Warn and ignore the object.            2: Error.            Backward: No interaction with Layout Error On Input.            RDB supported for severities 0 and 1.            RDB output is the merged expansion by 4 dbu of two edges forming the acute angle.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>POLYGON_AREA_ZERO</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A zero-area boundary exists in a GDSII-type database or a zero-area polygon exists in an OASIS database.            Severities:            0: Ignore the object.  <u>1</u>: Warn and ignore the object. (default)            2: Error.            Backward: No interaction with Layout Error On Input.            RDB supported for severities 0 and 1.            RDB output is the object itself.</p>
<p>Exception Name: <b>POLYGON_DEGENERATE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS, BINARY, ASCII            Description: A polygon has 0 or 1 vertices.            Severities:            0: Ignore the polygon.            1: Warn and ignore the polygon.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.            RDB supported for severities 0 and 1.            Zero-vertex objects are not output. Output for a single vertex object consists of a 4-dbu box around the vertex.</p>
<p>Exception Name: <b>POLYGON_IS_RECTANGLE</b>            Layout System: GDSII            Description: A two-point GDSII BOUNDARY is encountered.            Severities:  <u>0</u>: Interpret them as rectangles orthogonal to the database axes with no message. (default)            1: Interpret them as rectangles orthogonal to the database axes with a warning for each instance.            2: Error.            Backward: No interaction with Layout Error On Input.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>POLYGON_NONORIENTABLE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS, BINARY, ASCII            Description: A polygon cannot be oriented.            Severities:            0: Ignore the polygon.            1: Warn and ignore the polygon.  <u>2</u>: Error. (default)            3: Quietly process non-orientable polygons, if possible.            4: Warn and process non-orientable polygons, if possible.            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.            RDB support for severities 0, 1, 3, and 4.            RDB output is merged expansion by 4 dbu of the polygon edges.</p>
<p>Exception Name: <b>POLYGON_NONSIMPLE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS, BINARY, ASCII            Description: A polygon overlaps itself.            Severities:            0: Ignore the condition since the overlap is merged away. (default)            1: Warn and ignore the condition since the overlap is merged away.  <u>2</u>: Error.            Backward: No interaction with Layout Error On Input. <a href="#">Flag Nonsimple [Polygon]</a> YES sets severity 1.            RDB support for severities 0 and 1.            RDB output is the polygon itself.</p>
<p>Exception Name: <b>POLYGON_NOT_CLOSED</b>            Layout System: GDSII            Description: Controls how Calibre interprets open GDSII BOUNDARY parameters.            Severities:            0: Automatically close an open BOUNDARY with no message.            (default)            1: Automatically close an open BOUNDARY with a warning for each instance.  <u>2</u>: Error.            Backward: No interaction with Layout Error On Input.            RDB supported for severities 0 and 1. Output is the non-closed polygon.</p>

**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>RECTANGLE_SIDE_ZERO</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A rectangle (including a two-point GDSII polygon) has zero width or length.            Severities:            0: Ignore the rectangle.  <u>1</u>: Warn and ignore the rectangle. (default)            2: Error.            Backward: No interaction with Layout Error On Input.            RDB support for severities 0 and 1.            RDB output is a <math>4 \times 4</math> dbu square having a center point on the object.</p>
<b>Records</b>
<p>Exception Name: <b>BOX_RECORD</b>            Layout System: GDSII            Description: Error in handling BOX record for input layout database.            Severities:            0: Quietly ignore the record. (default)            1: Warn and ignore the record.            2: Quietly process the record. (default if <a href="#">Layout Process Box Record YES</a> is present)            3: Warn and process the record.            4: Fatal error.            Backward: No interaction with Layout Error On Input.            The Layout Process Box Record YES specification statement sets the severity to 2 only if the exception is not explicitly set with the Layout Input Exception Severity statement.            RDB support for severities 0, 1, 2, and 3.            RDB output is the polygon itself.</p>
<p>Exception Name: <b>CELLNAME_NSTRING</b>            Layout System: OASIS            Description: The string in a CELLNAME, CELL, or PLACEMENT record is not an OASIS n-string.            Severities:  <u>2</u>: Error. (default)            Backward: No interaction with Layout Error On Input.</p>

**Table 4-15. Exceptions (cont.)**

<p><b>Exception Name: NODE_RECORD</b></p> <p>Layout System: GDSII</p> <p>Description: Error in handling NODE record for input layout database.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li><u>0</u>: Quietly ignore the record. (default)</li> <li>1: Warn and ignore the record.</li> <li>2: Quietly process the record. (default if <a href="#">Layout Process Node Record YES</a> is present)</li> <li>3: Warn and process the record.</li> <li>4: Fatal error.</li> </ul> <p>Backward: No interaction with Layout Error On Input.</p> <p>The Layout Process Node Record YES specification statement sets the severity to 2 only if the exception is not explicitly set with the Layout Input Exception Severity statement.</p> <p>RDB support for severities 0, 1, 2, and 3.</p> <p>RDB output is the polygon itself.</p>
<p><b>Exception Name: PROPNAMENSTRING</b></p> <p>Layout System: OASIS</p> <p>Description: The string in a PROPNAMENSTRING or PROPERTY record is not an OASIS n-string.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li><u>2</u>: Error. (default)</li> </ul> <p>Backward: No interaction with Layout Error On Input.</p>
<p><b>Exception Name: TEXTSTRING_ASTRING</b></p> <p>Layout System: OASIS</p> <p>Description: The string in a TEXTSTRING or TEXT record is not an OASIS a-string.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li><u>2</u>: Error. (default)</li> </ul> <p>Backward: No interaction with Layout Error On Input.</p>
<p><b>Exception Name: XELEMENT_UNSUPPORTED</b></p> <p>Layout System: OASIS</p> <p>Description: An XELEMENT record (not supported) has been encountered.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li>0: Ignore the record.</li> <li>1: Warn and ignore the record.</li> <li><u>2</u>: Error. (default)</li> </ul> <p>Backward: No interaction with Layout Error On Input.</p>
<p><b>Exception Name: XNAME_UNSUPPORTED</b></p> <p>Layout System: OASIS</p> <p>Description: An XNAME record (not supported) has been encountered.</p> <p>Severities:</p> <ul style="list-style-type: none"> <li>0: Ignore the record.</li> <li>1: Warn and ignore the record.</li> <li><u>2</u>: Error. (default)</li> </ul> <p>Backward: No interaction with Layout Error On Input.</p>

**Table 4-15. Exceptions (cont.)**

Texts
<p>Exception Name: <b>LAYOUT_TEXT</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: A <a href="#">Layout Text</a> object has not been added since its corresponding cell name is not located in the input layout database.            Severities:            0: Ignore the object.            1: Warn and ignore the object. (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>
<p>Exception Name: <b>TEXT_BIG</b>            Layout System: GDSII            Description: A GDSII text object has more than 1 vertex.            Severities:            0: Ignore the object.            1: Warn and ignore the object.            2: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
<p>Exception Name: <b>TEXT_DEGENERATE</b>            Layout System: GDSII            Description: A GDSII text object has 0 vertices.            Severities:            0: Ignore the object.            1: Warn and ignore the object.            2: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
Units
<p>Exception Name: <b>METRIC_INPUT_FILE</b>            Layout System: GDSII, LEF/DEF, MILKYWAY, OPENACCESS            Description: In a multiple-file input layout database, the first input file has a different unit length than subsequent input files.            Severities:            0: Ignore the condition since the physical precision is not used anyway.            1: Warn and ignore the condition since the physical precision is not used anyway.            2: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>

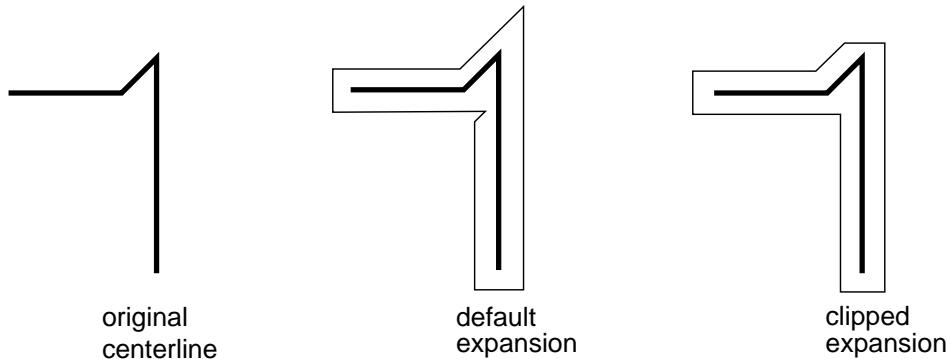
**Table 4-15. Exceptions (cont.)**

<p>Exception Name: <b>METRIC_RULE_FILE</b>            Layout System: GDSII, LEF/DEF, MILKYWAY, OPENACCESS            Description: The rule file <a href="#">Unit Length</a> does not agree with the physical precision in an input layout database file.            Severities:            0: Ignore the condition since the physical precision is not used anyway.  <u>1</u>: Warn and ignore the condition because the physical precision is not used anyway.            (default)            2: Error.            Backward: No interaction with Layout Error On Input.</p>
<p>Exception Name: <b>PRECISION_INPUT_FILE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: In a multiple-file input layout database, two files F1 and F2 have different database precisions.            Severities:            0: Ignore the condition because the rule file <a href="#">Precision</a> is used anyway.  <u>1</u>: Warn and ignore the condition.  <u>2</u>: Error. (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>
<p>Exception Name: <b>PRECISION_LAYOUT</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: The precision specified with <a href="#">Layout Precision</a> or the PREC keyword does not agree with the database precision in an input layout database file.            Severities:            0: Quietly ignore any differences.  <u>1</u>: Warn for each input layout database file whose precision is different from the Layout Precision value. (default)            2: Fatal error.            Backward: No interaction with Layout Error on Input.</p>
<p>Exception Name: <b>PRECISION_RULE_FILE</b>            Layout System: GDSII, OASIS, LEF/DEF, MILKYWAY, OPENACCESS            Description: The rule file precision does not agree with the database precision in an input layout database file.            Severities:            0: Quietly use the rule file <a href="#">Precision</a> and ignore the GDSII precision.  <u>1</u>: Warn and use the rule file precision.  <u>2</u>: Error (default)            Backward: If not explicitly set with Layout Input Exception Severity, then Layout Error On Input YES sets severity 2; NO sets severity 1.</p>

## Notes on Particular Exceptions

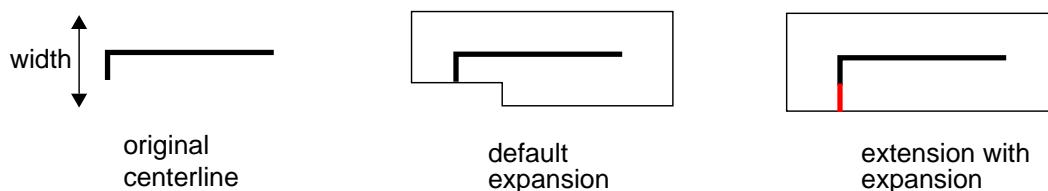
**PATH\_ACUTE** — The default algorithm for path expansion in GDSII- or OASIS-type input layout databases can, in certain cases, produce undesirable notches in paths with small acute angle jogs in their centerlines. This can be mitigated somewhat by replacing the acute angle by a 1dbu edge prior to expansion. Angles that are less than or equal to 45 degrees are clipped if one edge is orthogonal. See [Figure 4-144](#). This behavior is controlled with the PATH\_ACUTE exception.

**Figure 4-144. Acute Angle Path Expansion**



**PATH\_ENDSEGMENT\_SHORT** — The default algorithm for path expansion in GDSII- or OASIS-type input layout databases produces a notch at the end of paths where the end segment is less than half the path width. This can be mitigated by extending the end segment to the half-width as shown [Figure 4-145](#).

**Figure 4-145. Path Expansion for Short End Segments**



## Examples

### Example 1

```
// Use rule file precision if it disagrees with layout DB precision
LAYOUT INPUT EXCEPTION SEVERITY precision_rule_file 1
```

### Example 2

```
LAYOUT INPUT EXCEPTION SEVERITY path_angled 2 RDB
// warn but ignore angled paths. output to RDB.

LAYOUT INPUT EXCEPTION RDB exceptions.rdb
```

# Layout Magnify

Specification statement

**LAYOUT MAGNIFY { {*value* [PLACE]} | AUTO }**

Used only in Calibre.

## Parameters

- ***value***  
A positive floating-point number or numeric expression, which can include variables (see [Variable](#)). This parameter or **AUTO** must be specified.
- **PLACE**  
Optional keyword that specifies a simulated cell containing the entire input layout database (except the actual top-level cell) has been placed in the top cell at the origin and at the given magnification.
- **AUTO**  
Keyword that specifies that the magnification value is calculated automatically. The magnification is the ratio of the value of the [Precision](#) specification statement over the physical precision of the input layout database.

## Description

Specifies that the input layout database, as it is being read into the Calibre application, is magnified by the given ***value***.

If the **AUTO** keyword is specified, then the magnification is calculated to be the rule file Precision over the database precision. If this statement appears in your rule file, the transcript reports this prior to the input database being read.

The magnification algorithm multiplies all coordinate data, including placement base points (for GDSII and OASIS) and array pitches (for GDSII), by the given ***value*** and rounds all values to integers, regardless of hierarchical position. This algorithm is equivalent, disregarding mathematical round-off error, to placing the entire input layout database at the given magnification into a new top-level cell; the difference is that the new cell is not explicitly created.

[Layout Polygon](#), [Polygon](#), [Layout Text](#), and [Text](#) statement objects behave as follows when Layout Magnify is used. When ***value*** is specified, the objects have the magnification factor applied to their coordinates. When **AUTO** is specified, the objects do not have magnification applied to them.

When using the PLACE argument, multiplication does not occur as the database is being read. Instead, Calibre simulates that a cell containing the entire input layout database (except the actual top-level cell) has been placed in the top cell at the origin and at the given magnification. For flat applications, this is equivalent to multiplying coordinate data by the magnification ***value*** after database flattening. For hierarchical applications, the simulated cell triggers the

algorithm used to remove magnified instances during the hierarchical database construction phase. The simulated cell is then removed from the hierarchy when this algorithm completes.

The PLACE algorithm is no more or less accurate than the default behavior, so the one you use is purely a matter of preference. However, if writing out mask data at a magnification *value* that is less than 1, the PLACE option is recommended to reduce gaps that occur due to grid snapping.

This statement may be specified once in your rule file. Magnification can also be specified for individual files using the [Layout Path](#) and [Layout Path2](#) statements.

Layout Magnify is often used when increasing measurement precision. The rule file [Precision](#) setting should be modified to accommodate such uses of Layout Magnify. See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

See also [DRC Magnify Results](#), [DRC Magnify Density](#), [DRC Magnify NAR](#), and [Magnify](#).

## Examples

Whenever layout database precision is altered, you should specify the [Layout Precision](#) statement or [Layout Input Exception Severity](#) PRECISION\_RULE\_FILE 1 or 0. Redefinition of layout precision introduces an implied scaling of the data, which must be compensated for by a Layout Magnify statement. To avoid possible rounding errors, the product of the Layout Magnify and [DRC Magnify Results](#) values should produce an integer, or the magnification value should have a *greater number* of significant figures than the number of significant figures for any coordinate in the design. For all of these examples, “dbu” means database units.

### Example 1

To rescale an input database from 1000 dbu/um to 10000 dbu/um, and to have the result at 100% original size:

```
PRECISION 10000      //scale down by factor of 10
LAYOUT PRECISION 1000 //specify the expected database precision
LAYOUT MAGNIFY AUTO  //automatically magnify by factor of 10000/1000
```

### Example 2

To rescale an input database from 1000 dbu/um to 10000 dbu/um, and to have the result at 50% of original size:

```
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 1
PRECISION 10000 //scale down by factor of 10
LAYOUT MAGNIFY 5 //magnify by factor of 5
```

The previous method is preferred to simply doing this:

```
LAYOUT MAGNIFY 0.5 //reduce by 50%
```

It is generally recommended that the output precision be increased by a factor of 10 when the magnification is less than 1, such as in Example 2, to reduce the number of gaps that can be introduced due to grid snapping.

### Example 4

To rescale an input database from 1000 dbu/um to 10000 dbu/um, and to have the result at 42% of original size with DRC results at original scale:

```
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 1
PRECISION 10000 //scale down by factor of 10
LAYOUT MAGNIFY 4.2 //magnify by factor of 4.2

DRC RESULTS DATABASE PRECISION 1000
/* scale results up by a factor of 10 */
DRC MAGNIFY RESULTS 0.238095
```

The DRC Magnify Results value is set to this:

output precision / physical precision \* (1 / layout magnification)

in this case:

$$1000/1000 * (1/4.2) = 0.2380952$$

This is to reduce the amount of grid snapping that occurs. You should truncate the DRC Magnify Results value at a number of significant figures that is *greater than* the maximum number of significant figures for any coordinate in the design. For example, if the coordinate 550.175 has the maximum number of significant figures in the design, then the DRC Magnify Results value should have at least seven significant figures. This is to reduce rounding and grid snapping problems.

## Layout Merge On Input

Specification statement

**LAYOUT MERGE ON INPUT {NO | YES | {*layer* [*layer...*]}}**

Used only in Calibre.

### Parameters

- **NO**  
Specifies that layout shapes (typically from module generators) are not merged on a per-cell, per-layer basis on input. This is the default.
- **YES**  
Specifies that layout shapes (typically from module generators) are merged on input.
- ***layer***  
Specifies that only the specified layer is merged on input. This parameter can be an original layer name or number. If a layer set name is used, then this is equivalent to all the constituent layers of the set being specified. This parameter may be specified more than once.

### Description

Controls layout database layer merging. This statement may only be specified once if NO or YES are used. Otherwise, it may appear multiple times.

The NO keyword directs the layout reader not to merge shapes upon read-in. Database merging occurs on a layer-by-layer basis using single-layer OR statements after the database is read in.

The YES keyword directs the layout reader to merge shapes on a per-cell, per-layer basis as it reads the data stream into memory. Generally, you would use this only for databases created by module generator tools that redundantly place shapes on top of each other. A layout database that originated from such a module generation tool can contain many overlapping shapes on a single layer within each cell. Merging reduces this number dramatically. It is often desirable to merge on a per-layer, per-cell basis prior to merging the flat representation, which the verification system does automatically for essentially every original layer.

In cases such as this, you reduce the time for merging of the flattened layers and increase the time to read the data stream. *Do not* specify YES, however, if the database lacks the module generator characteristics.

For example, assume that on layer L in cell C there are N shapes. When merged there would be M shapes, where M is much less than N. In addition, if cell C has P flat placements, the total number of shapes resulting from the flattening of layer L from cell C is  $N \times P$ . This reduces the number of shapes considerably, to  $M \times P$ , if P is large and M is much less than N. Merging the flat layer saves memory and makes the run more efficient.

The YES option is not as useful for Calibre nmDRC-H because the number of flat placements does not matter in that context.

## Layout Merge On Input

---

Polygon flagging (for example, using the Flag family of specification statements, or the Drawn family of layer operations) might occur differently if you select YES. The Drawn and Flag families of commands operate on shapes immediately after database read-in and before the single-layer OR operations occur. If the layout has been merged on read-in (which occurs when YES is specified), then overlapping shapes can be merged together when the Drawn or Flag command analyzes the shapes.

Specifying YES has no effect on binary or ASCII input database formats.

If you specify a *layer* parameter, then only the specified layer is merged on a per-cell, per-layer basis. More than one *layer* may be specified. This capability can be useful in advanced *methodology-type* DRC checks. For example, assume this rule:

Shapes on layer POLY may not have inter-hierarchical overlaps but may overlap in the same cell. On the other hand, shapes on layer DIFF may not overlap anywhere in the hierarchy.

Solution:

```
LAYOUT MERGE ON INPUT poly
...
rule {
  AND poly
  AND diff
}
```

## Examples

```
LAYOUT MERGE ON INPUT NO
```

# Layout Path

Specification statement

**LAYOUT PATH {*filename* [*file\_options*] [*filename* [*file\_options*] ]... } | STDIN**

Used only in Calibre.

## Summary

Specifies the pathnames of layout design files. This statement is required for any tool that reads a layout database.

## Parameters

- *filename*

A pathname of the layout database. You can specify *filename* any number of times in one statement. Filenames may appear in quotation marks (""). If you do not specify a filename, then you must specify **STDIN**.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

- *file\_options*

Optional parameters that control the processing of the *filename* layout database. The syntax of the options is:

[MAG {*number* | AUTO}] [PREC {*value* | { *valueD valueN* } }] [GOLDEN]

- MAG {*number* | AUTO}

Optional keyword that applies a magnification factor to the corresponding *filename* database.

*number* — A positive floating-point value (interpreted as dimensionless) that may also be a numeric expression. The *number* is the magnification factor used.

AUTO — Specifies that the magnification value is calculated automatically. The magnification is the ratio of the value of the [Precision](#) specification statement over the physical precision of the input layout database.

If MAG is specified for a *filename*, then the value given here overrides the [Layout Magnify](#) statement.

The MAG keyword is not supported with LEF/DEF layout databases.

- PREC {*value* | { *valueD valueN* } }

Optional keyword that specifies an input layout database precision for the corresponding *filename*. This value is used for checking the input database precision, and has similar semantics as described in [Layout Precision](#).

*value* — A positive floating-point numeric value defining the expected database precision value.

*valueD valueN* — Positive integer values that define, respectively, the numerator and denominator of a ratio. The expected database precision value is  $valueN/valueD$ .

If PREC is specified, then the value given overrides both the [Precision](#) and [Layout Precision](#) statements, if present. This is for the purpose of checking the input database precision only. It does not affect the precision of measurements made during the run.

- **GOLDEN**

Optional keyword that specifies [Extent Cell](#), [Inside Cell](#), or [Not Inside Cell](#) ... WITH MATCH GOLDEN operations attempt to match “golden” cells only from the Layout Path statement that also uses GOLDEN. This keyword should only be used for hierarchical runs.

- **STDIN**

Keyword that specifies the layout comes from standard input.

## Description

Specifies the layout database pathname(s) for the database type specified by the [Layout System](#) statement. Can be specified any number of times with multiple pathnames to the layout files for all database types except LEF/DEF. This statement is required in Calibre.

The *filename*(s) specified should conform to the following:

- For GDSII, OASIS, and SPICE files, *filename* refers to the respective layout file.
- For MILKYWAY and OPENACCESS files, *filename* refers to the respective design database. For the OPENACCESS system, your OA library pathname must be specified in your lib.defs file. Calibre uses the lib.defs file it finds in this search order: current working directory, \$HOME directory, or <OA\_install\_path>/data/lib.defs.
- For CNET databases, *filename* refers to a CNET database directory (flat nmLVS applications only).
- For LEF/DEF files, at least two filenames (at least one for LEF and only one for DEF) must be present. All LEF entries must be specified before the DEF entry.

The LEF filenames specify either individual files, or one directory containing all the LEF files. When specifying a directory, all LEF (including TLEF) files must be in the same directory.

In Calibre xRC, DEF entry may specify either a single DEF file or a directory that contains all the DEF files.

The notation used in xRC is different from other Calibre applications, as described here:

**Calibre applications other than xRC** — multiple LEF files must all appear within a single set of quotation marks (“ ”). For example:

```
// non-xRC application
LAYOUT PATH "cmos_tech.lef cmos_ams.lef" top.def
```

**Calibre xRC** — no special notation is needed for multiple LEF files. For example:

```
// xRC application
LAYOUT PATH cmos_tech.lef cmos_ams.lef top.def
```

The MAG keyword is not supported with the LEF/DEF databases.

- For ASCII and BINARY formats this statement is ignored and the current directory is assumed.

In layout systems GDSII, LEFDEF, and OASIS, multiple Layout Path *filename* parameters indicate a concatenation of files. In layout systems MILKYWAY, OPENACCESS, SPICE, and CNET, only the first *filename* from the first Layout Path statement is used.

In layout systems GDSII and OASIS, multiple input databases specified in a Layout Path statement are treated as if all the structure records (for GDSII) or cell records (for OASIS) were embedded in the first file specified. Each input database file, however, is expected to be syntactically complete. Whether or not multiple files are specified for the input layout database, multiple records for the same layout cell are not allowed by default. All identically-named records after the first are discarded (with a warning or error). You may control this behavior with the [Layout Allow Duplicate Cell](#) specification statement. The default is NO.

For geometric databases, the physical precision (the grid of the database) must match the [Precision](#) statement in the rule file or a fatal error results. To permit differing database and rule file precisions, use the [Layout Precision](#) statement or [Layout Input Exception Severity](#) PRECISION\_RULE\_FILE with a setting of 0 or 1. Input of two or more databases with differing database precisions is a fatal error by default. The PREC keyword allows input of databases with differing precisions.

If the MAG keyword is specified after a *filename*, the magnification algorithm multiplies and rounds all coordinate data (including placement base points and array pitches for GDSII and OASIS) by the given value as the data is being read, regardless of its hierarchical position. This algorithm is identical to specifying the [Layout Magnify](#) statement without the PLACE keyword. The MAG keyword overrides a Layout Magnify statement for the specified *filename*.

---

**Note**

A rule file compilation error occurs if Layout Magnify is specified with the PLACE argument and MAG is specified with Layout Path.

---

Layout magnification is often used to increase database precision. If you perform layout magnification, you should consider changing your rule file [Precision](#) accordingly. See “[Layout](#)

[Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

If the GOLDEN keyword is specified after a *filename*, this does the following:

- Designates all cells in the given layout database input file as “golden” for the purpose of Exclude Cell or [Not] Inside Cell ... WITH MATCH GOLDEN operations.
- Avoids placing these “golden” cells into the true design hierarchy (including as the top-level cell).
- Allows these “golden” cells’ names to duplicate those of cells in a Layout Path specification statement without GOLDEN (no errors or concatenation of contents).

Layout databases specified with the GOLDEN keyword should not contain cells that match by name (that is, golden cells are assumed to be unique). If such duplication occurs, a runtime error is issued.

## Compressed Files

Any GDSII, OASIS, or SPICE *filename* ending in .gz or .Z is treated as a compressed file. Such files can be opened automatically by Calibre using gzip or uncompress, respectively, if these utilities are in your environment. Some versions of these utilities cannot open files larger than two GB. If you cannot open compressed files larger than two GB, you may need to update to a newer version of gunzip or uncompress. These utilities may also have difficulty opening files across a network. Neither of these issues are Calibre limitations.

If compressed files are in SPICE format, these files can be either libraries named in a .LIB statement or included files named in a .INCLUDE statement. Due to the limitations of the technology of uncompression routines, it is not advisable to compress files that have .INCLUDE statements or .LIB statements that are not placed near the top of the files that reference, rather than define, a library. This is because the compressed file must always be read from the beginning when returning from processing the included file or the referenced library. For example, it is not advisable to compress files that look like this:

```
$ Long SPICE file:  
...  
... many lines of SPICE  
...  
$ Followed by .INCLUDE:  
.INCLUDE somefile  
$ Reorder this file or you may experience performance issues
```

It is preferable to reorder the file so the .INCLUDE statement appears near the top of the file.

## Command Line Override for nmLVS

The -layout *spice\_file\_name* command line option in nmLVS overrides the Layout Path and [Layout System](#) statements in the rule file. If your Layout Path and Layout System specify geometric layouts such as GDS or OASIS, the -layout option allows you to override these settings with the path to a SPICE file.

Layout Path has no effect on ICVerify applications.

In Calibre Interactive, the Layout Path is specified from the **Inputs > Layout** tab. See the *Calibre Interactive and Calibre RVE User's Manual* for details.

See also [Layout Primary](#), [Layout System](#), [Layout Path2](#), [Source Path](#).

## Examples

### Example 1

SPICE-to-SPICE comparison.

```
LAYOUT PATH "/tmp/layout.spi"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM SPICE

SOURCE PATH "/home/spice_sources/mem.spi"
SOURCE PRIMARY "memory_block"
SOURCE SYSTEM SPICE
```

### Example 2

LEFDEF layout applications, for a non-xRC application.

```
// non-xRC application
LAYOUT PATH "lef_file1.lef lef_file2.lef" file.def
LAYOUT PRIMARY TOP
LAYOUT SYSTEM LEFDEF
```

### Example 3

Concatenating layout files.

```
LAYOUT PATH "/home/layouts/base.oas" "/home/layouts/top.oas"
LAYOUT PRIMARY TOP
LAYOUT SYSTEM OASIS
```

### Example 4

This example re-scales the measurement precision from 1000 dbu/um to 10000 dbu/um using the rule file Precision statement. It then uses Layout Path with the PREC keyword to check the database precision and the MAG AUTO keyword set to scale the database at 100% original size.

```
PRECISION 10000      //scale down by factor of 10
//check database precision at 1000 and magnify to original scale
LAYOUT PATH "/home/layouts/base.oas" PREC 1000 MAG AUTO
```

### Example 5

This example shows the use of the GOLDEN option. Any cells specified in [Extent Cell](#), [Inside Cell](#), or [Not Inside Cell](#) ... WITH MATCH GOLDEN layer operations are expected to match cell names in lib.gds. However, these cell names do not necessarily match any names in layout.gds. Such cells are not concatenated with layout.gds. A geometric match is attempted between cells in WITH MATCH GOLDEN operations and cells in layout.gds.

```
LAYOUT PATH "./layout.gds" // main design; cell name correspondences with
                           // lib.gds are not necessarily known.
LAYOUT PATH "./lib.gds" GOLDEN      // cell library with golden cells

...
// EXTENT CELL or [NOT] INSIDE CELL operations using WITH MATCH GOLDEN.
// Cell names match cells in lib.gds, but do not necessarily match
// layout.gds.
...
```

## Layout Path2

Specification statement

**LAYOUT PATH2** *filename* [*file\_options*] [*filename* [*file\_options*] ... ]

Used only in Calibre.

### Summary

Specifies a second database when performing dual-database applications. Dual database comparison is described under “[Layout Versus Layout Comparison](#)” in the *Calibre Solutions for Physical Verification* manual and “[Dual Database Comparison](#)” in the *Calibre Verification User’s Manual*.

### Parameters

- ***filename***

A required filename of the second layout database for dual-database applications. You can specify *filename* any number of times in one statement. GDSII, OASIS, LEF/DEF, and OpenAccess databases are supported.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

- ***file\_options***

Optional parameters that control the processing of the *filename* layout database. The syntax of the options is:

[MAG {*number* | AUTO}] [PREC {*value* | { *valueD valueN* } }] [GOLDEN]

- MAG {*number* | AUTO}

Optional keyword that applies a magnification factor to the corresponding *filename* database.

*number* — A positive floating-point value (interpreted as dimensionless) that may also be a numeric expression. The *number* is the magnification factor used.

AUTO — Specifies that the magnification value is calculated automatically. The magnification is the ratio of the value of the [Precision](#) specification statement over the physical precision of the input layout database.

If MAG is specified for a *filename*, then the value given here overrides the [Layout Magnify](#) statement.

- PREC {*value* | { *valueD valueN* } }

Optional keyword that specifies an input layout database precision for the corresponding *filename*. This value is used in checking the input database precision, and has the same semantics as described in [Layout Precision](#).

*value* — A positive floating-point numeric value defining the expected database precision value.

*valueD valueN* — Positive integer values that define, respectively, the numerator and denominator of a ratio. The expected database precision value is  $\text{valueN}/\text{valueD}$ .

If PREC is given for a *filename*, then the value given here overrides both the [Precision](#) and [Layout Precision](#) statements, if present. This is for the purpose of checking the input database precision only. It does not affect the precision of measurements made during the run.

- GOLDEN

Optional keyword that specifies [Extent Cell](#), [Inside Cell](#), or [Not Inside Cell](#) ...  
WITH MATCH GOLDEN operations attempt to match “golden” cells only from the Layout Path2 statement that also uses GOLDEN.

### Description

Specifies the second layout database filename(s) for dual-database applications in Calibre. The database specified by the *filename* argument is read in along with the database specified in the [Layout Path](#) statement. This statement can be specified any number of times; multiple files are concatenated into a single hierarchy maintained separately from the Layout Path design hierarchy.

If you specify this statement, then you must also specify [Layout System2](#) and [Layout Primary2](#) statements.

The *filename*(s) specified should conform to the following:

- For GDSII and OASIS files, *filename* refers to the respective layout file.
- For OPENACCESS files, *filename* refers to the respective design database. Your OA library pathname must be specified in your lib.defs file. Calibre uses the lib.defs file it finds in this search order: current working directory, \$HOME directory, or <OA\_install\_path>/data/lib.defs.
- For LEF/DEF files, at least two filenames (at least one for LEF and only one for DEF) must be present. All LEF entries must be specified before the DEF entry.

The LEF filenames specify either individual files, or one directory containing all the LEF files. When specifying a directory, all LEF (including TLEF) files must be in the same directory.

If the MAG keyword is specified after a *filename*, the magnification algorithm multiplies and rounds all coordinate data (including placement base points and array pitches for GDSII and OASIS) by the given value as the data is being read, regardless of its hierarchical position. This algorithm is identical to specifying the [Layout Magnify](#) statement without the PLACE option. The MAG keyword overrides a Layout Magnify statement for the specified *filename*.

**Note**

A rule file compilation error will result if Layout Magnify is specified with the PLACE argument and MAG is specified with Layout Path2.

Layout magnification is often used to increase database precision. If you perform layout magnification, you should consider changing your rule file [Precision](#) accordingly. See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

If the GOLDEN keyword is specified after a *filename*, this does the following:

- Designates all cells in the given layout database input file as “golden” for the purpose of Exclude Cell or [Not] Inside Cell ... WITH MATCH GOLDEN operations.
- Avoids placing these “golden” cells into the true design hierarchy (including as the top-level cell).
- Allows these “golden” cells’ names to duplicate those of cells in a Layout Path2 specification statement without GOLDEN (no errors or concatenation of contents).

Note that if a cell appears in a path specified by Layout Path2 GOLDEN, but there is no corresponding layer operation using the WITH MATCH GOLDEN keywords for that cell, the tool will attempt to concatenate that cell with other Layout Path2 databases in the flow. This situation can cause duplicate cell errors.

In layout systems GDSII and OASIS, multiple input databases specified in a Layout Path2 statement are treated as if all the structure records (for GDSII) or cell records (for OASIS) were embedded in the first file specified. Each input database file, however, is expected to be syntactically complete. Whether or not multiple files are specified for the input layout database, multiple records for the same layout cell are not allowed by default. All identically-named records after the first are discarded (with a warning or error). You may control this behavior with the [Layout Allow Duplicate Cell](#) specification statement. The default is NO.

**Compressed files** — Any filename ending in .gz or .Z is treated as a compressed file. Such files can be opened by Calibre using gunzip or uncompress, respectively, if these utilities are in your PATH variable. Some versions of these utilities cannot open files larger than two GB. If you cannot open compressed files larger than two GB, you may need to update to a newer version of gunzip or uncompress. These utilities may also have difficulty opening files across a network. Neither of these issues are Calibre limitations.

See also [Layout Bump2](#).

## Examples

```
//SPECIFY FIRST DATABASE
LAYOUT PATH "/tmp/ring_new.gds"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM GDSII
```

## Layout Path2

---

```
//SPECIFY SECOND DATABASE
LAYOUT PATH2 "/tmp/ring.gds"
LAYOUT PRIMARY2 "TOP"
LAYOUT SYSTEM2 GDSII
LAYOUT BUMP2 200 // add 200 to layer numbers for this database

...
compare_1 { 1 XOR 201 }
compare_2 { 2 XOR 201 }
```

## Layout Place Cell

Specification statement

LAYOUT PLACE CELL *filename* [ GDSII | GDS | GDS2 | OASIS ] *top\_cell placed\_cell\_info*  
*[placed\_cell\_info ...]*

Used only in Calibre.

### Parameters

- *filename*

Required parameter giving the filename for the constructed database.

- GDSII | GDS | GDS2 | OASIS

Optional parameter giving the database type. GDSII is the default.

- *top\_cell*

A required argument that gives the name of the top cell in the created database. Only one cell record is created.

- *placed\_cell\_info*

A required parameter that gives the information necessary to instantiate a cell placement. The parameter *placed\_cell\_info* has the following syntax:

*cell\_name magnification angle reflection x y*

with these parameter definitions:

***cell\_name*** — The name of the instantiated cell. Cells referenced by this parameter must be defined in an input database.

***magnification*** — An undimensioned positive floating-point number, which gives the magnification. It may be a variable.

***angle*** — An undimensioned floating-point number, which gives the angular rotation in degrees of the cell placement. It may be a variable.

***reflection*** — An integer constant that must be either 0 or 1, with 1 meaning x-axis reflection and 0 meaning no reflection.

***x y*** — Dimensioned floating-point numbers that give the coordinates of the cell placement origin in user units. They may be variables. Negative numbers must appear in parentheses ( ).

### Description

Specifies construction of a GDSII or OASIS database within the Calibre application.

Calibre applications generally begin by compiling the rule file. After compilation, the application determines if a Layout Place Cell specification statement is present. If present, Calibre immediately constructs a GDSII or OASIS database with the given *filename*. Only one cell record (BGNSTR in GDSII) is created, and the name of the cell is *top\_cell*. The database

precision is that given in the rule file [Precision](#). For GDSII databases, the LIBNAME is “drc.db”.

Each **placed\_cell\_info** argument instantiates a placement (SREF in GDSII) of a cell having the **cell\_name** within **top\_cell**. The cell given by a **cell\_name** parameter appears in some other database specified in the Layout Path. The placement transform is given by the **magnification**, the **angle**, the **reflection** and the **(x, y)** location of the cell origin.

The usage model is normally that the [Layout Path](#) and [Layout Primary](#) specification statements reference this new database. The [Layout System](#) is specified accordingly.

### Examples

```
LAYOUT PATH new.gds A.gds B.gds      // A.gds contains cell A
                                         // B.gds contains cell B
LAYOUT PRIMARY new_top
LAYOUT SYSTEM GDSII
...
// This is our new input layout database.
LAYOUT PLACE CELL new.gds GDS new_top
    A 1.5 90 0 65.3 89.7 // Places cell A
    B 1.0 270 1 987.3 0   // Places cell B
```

## Layout Polygon

Specification statement

**LAYOUT POLYGON** *vertex\_count* *x1 y1 x2 y2* [*xN yN ...*] *layer* [*datatype*] [*cell\_name*]

Used only in Calibre.

### Parameters

- *vertex\_count*

A required integer greater than or equal to 2 that specifies the number of user-defined vertices for a polygon. It must match the number of (x, y) coordinate pairs specified in the statement.

- *x1 y1 x2 y2* [*xN yN ...*]

A required set of floating-point coordinates that represent the vertices of the polygon in user-units. At least two pair of coordinates are required. Exactly two pair of coordinates are interpreted as the lower-left and upper-right corners (they must be specified in that order) of a rectangle having sides parallel, respectively, to the coordinate axes. The number of (x, y) coordinate pairs must match the *vertex\_count*. These coordinates may be numeric variables, as specified in a [Variable](#) statement.

By default, the coordinates are top-level coordinates. If a *cell\_name* is specified that is not the primary cell, then the coordinates are interpreted as local to the specified cell.

- *layer*

The *layer* must be an original layer number or an original layer name. If *layer* is a simple layer name, this is equivalent to the usage of its layer number in the statement. If *layer* is a layer set name, this is equivalent to the statement being repeated for each element of the layer set. The layer does not have to be in the input layout database.

- *datatype*

An optional integer specifying the datatype of the polygon. The default is 0.

- *cell\_name*

An optional cell name that indicates a cell in which the polygon is to appear. Any wildcard characters in *cell\_name* are treated as literal. The default is the top-level cell name.

### Description

Allows original database geometry to be specified directly in the rule file. This statement defines a polygon having the given coordinates on the specified *layer*, the specified *datatype* (0 if omitted), and in the specified cell (primary cell if *cell\_name* is omitted). The polygons generated by this statement behave in most respects as if they were in the layout database.

This statement may only be used by Calibre applications if the [Layout System](#) is GDSII or OASIS. It may be specified any number of times.

## Layout Polygon

---

The first and last coordinates in the coordinate list need not coincide, and the polygon may be oriented either counter-clockwise or clockwise. The polygon must be simple; that is, it may not self-intersect.

If the *cell\_name* is specified, but does not exist in the input layout database, then a warning is issued and the polygon is ignored by default. This behavior can be altered based upon the value of the Layout Input Exception Severity parameter: LAYOUT\_POLYGON.

Vertex coordinates in user units generated using this statement are not affected by changes in the [Precision](#) setting. However, coordinates in database units do follow the Precision setting. For example, if you specify (1, 1) as the coordinates of a vertex and the Precision is 1000, the vertex is at (1000, 1000) in database units. If you change the Precision to 10000, the user coordinates remain at (1, 1) but the database coordinates become (10000, 10000).

Polygons generated by this statement are affected by [Layout Magnify](#) with an explicit *value*; the **AUTO** keyword has no effect, however. The [Layout Path](#)[2] MAG keyword has no effect on Layout Polygon.

The differences between this statement and the [Polygon](#) specification statement are as follows:

- Layout Polygon requires a vertex count to be specified; Polygon does not.
- Layout Polygon allows a datatype to be specified; Polygon does not.
- Layout Polygon allows a cell name to be specified; Polygon does not.
- Layout Polygon may not be used in ICVerify applications; Polygon may.

## Examples

```
//SPECIFY A METAL SQUARE IN THE "PAD" CELL  
LAYOUT POLYGON 2 1.0 -1.0 10.1 10.1 metal PAD
```

# Layout Precision

Specification statement

**LAYOUT PRECISION** {*value* | { *valueD valueN* } }

Used only in Calibre.

## Parameters

- ***value***  
A positive floating-point numeric value defining the expected database precision.
- ***valueD valueN***  
Positive integer values that define a ratio *valueN/valueD*, which is the expected database precision.

## Description

The Layout Precision statement allows specification of an alternate precision value (instead of the rule file [Precision](#) value) for checking the precision of input layout databases in the Calibre layout data input module.

This statement is optional and can appear once in a rule file. Note that Layout Precision is for checking purposes only. It has no bearing on any other Calibre semantics or operations.

When checking the precision of the input layout database, the database precision value given by Layout Precision overrides the value specified by [Precision](#).

If your database precision does not match the specified precision, an error results. You can override this error with the [Layout Input Exception Severity](#) PRECISION\_LAYOUT setting.

You can set the precision separately for each input layout database file using the PREC keyword in the [Layout Path](#) and [Layout Path2](#) statements.

See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

See also [Layout Use Database Precision](#) and [Layout Magnify](#).

## Example

### Example 1

Assume the following is specified:

```
LAYOUT PRECISION 2000
```

Then each input layout database specified in the Layout Path and Layout Path2 specification statements has its precision checked against the value 2000 instead of the Precision specification statement value. The Layout Path and Layout Path2 statements can override this value by using the PREC keyword.

### Example 2

To rescale an input database from 1000 dbu/um to 10000 dbu/um, and to have the result at 100% original size:

```
PRECISION 10000      //scale down by factor of 10
LAYOUT PRECISION 1000 //specify the expected database precision
LAYOUT MAGNIFY AUTO   //automatically magnify by factor of 10000/1000
```

# Layout Preserve Case

Specification Statement

**LAYOUT PRESERVE [NET | PROPERTY] CASE {NO | YES}**

## Parameters

- **NO**

Keyword that preserves the case of layout net names, but treats names that differ only by case as identical during connectivity extraction. This is the default behavior if this statement is not specified in the rule file. This keyword also causes **Device** property names to be written in lowercase whether or not PROPERTY is also specified.

- **YES**

Keyword that specifies layout net names must match exactly, including case, to be considered identical when NET is also specified, or if NET is accepted as the default. If PROPERTY is specified, then the case of Device property names is preserved when they are written.

- **NET**

Optional keyword that specifies the statement applies to layout net names. This is the default behavior. If specified, this keyword must appear as shown previously in the syntax line.

- **PROPERTY**

Optional keyword that specifies the statement applies only to Device property names. If specified, this keyword must appear as shown previously in the syntax line.

## Description

This statement controls how layout text names are handled during connectivity extraction. It also controls how Device properties are written to the extracted layout netlist, the SVDB, all results generated by the Query Server, and so forth.

Each NET or PROPERTY keyword may appear in at most one Layout Preserve Case statement. The following table shows how these keywords are handled:

**Table 4-16. Layout Preserve Case Keywords**

First keyword	Second Keyword	Effect on layout nets	Effect on properties
NET	NO	Net names that differ by case are considered identical during connectivity extraction. This is true whether this keyword pair is explicitly specified or is in effect by default.	Device property names are written in lowercase. This is true whether this keyword pair is explicitly specified or is in effect by default.

**Table 4-16. Layout Preserve Case Keywords (cont.)**

<b>First keyword</b>	<b>Second Keyword</b>	<b>Effect on layout nets</b>	<b>Effect on properties</b>
NET	YES	Net names that differ by case are considered different during connectivity extraction. This is true whether or not NET is used as the default keyword.	Device property names are written in lowercase, unless PROPERTY and YES are used in another statement.
PROPERTY	NO	No effect on layout nets.	Device property names are written in lowercase.
PROPERTY	YES	No effect on layout nets.	Device property names are written in the same case as what appears in the rule file.

Remember that all LVS comparison and matching of property names, such as property names in Trace Property statements, remain case-insensitive because Layout Preserve Case only affects how property names are written, not how they are matched. For example, consider the following rule file statements:

```
DEVICE R RES MET1 MET1 [ PROPERTY Val = 1 ]
LAYOUT PRESERVE PROPERTY CASE YES
TRACE PROPERTY RES val VAL 0
```

The extracted netlist has a property named “Val” on all resistors, for example:

```
R1 1 2 Val=1 $X=300 $Y=600 $D=1
```

The Trace Property statement causes LVS to consider this layout property “Val” equivalent with a source property “val”, because matching of property names is case-insensitive (although tracing of string properties can be made case-sensitive).

The Net and Not Net layer operations are case-sensitive when Layout Preserve [Net] Case YES is specified.

For a complete discussion of the selection of extracted layout netlist names, see “[Selection of Extracted Netlist Names](#)” in the *Calibre Verification User’s Manual*.

For case-sensitive netlisting of user-defined devices that have the same number of pins, see [Layout Case](#).

## Examples

### Example 1

In the following example, the same net is named “abc” in one location and “ABC” in another location. This is not considered a short circuit and the net name in the extracted SPICE netlist is either “abc” or “ABC,” chosen arbitrarily. Alternatively, the names “abc” and “ABC” appearing on different nets are treated as an open circuit, and one of the names is removed.

```
LAYOUT PRESERVE CASE NO
```

**Example 2**

In the following example, the same net is named “xyz” in one location and “XYZ” in another location, and is considered a short circuit. These same names appearing on two different nets are not considered an open circuit and both names appear in the extracted netlist.

```
LAYOUT PRESERVE CASE YES
```

**Example 3**

In this example, net names are treated in a case-insensitive manner during connectivity extraction, but the case of Device property names is preserved when they are written:

```
// Layout Preserve [NET] Case NO is in effect by default.  
LAYOUT PRESERVE PROPERTY CASE YES
```

This is the same as specifying either:

```
LAYOUT PRESERVE CASE NO // NET is assumed by default  
LAYOUT PRESERVE PROPERTY CASE YES
```

or:

```
LAYOUT PRESERVE NET CASE NO  
LAYOUT PRESERVE PROPERTY CASE YES
```

## Layout Preserve Cell List

Specification statement

### LAYOUT PRESERVE CELL LIST *list\_name*

Used only in hierarchical Calibre.

#### Parameters

- *list\_name*

A required name of a list of cells specified in a [Layout Cell List](#) specification statement.

#### Description

Preserves the specified layout cells from expansion during hierarchical runs. Cells specified in this statement are treated in the same manner as hcells in nmDRC-H and in nmLVS-H connectivity extraction.

The *list\_name* is a list specified in a Layout Cell List statement. It is possible to specify these cells using a wildcard or a regular expression in the Layout Cell List statement. Use of wildcards and regular expressions is discussed under [Layout Cell List](#).

Cells declared in the Layout Preserve Cell List statement have no relation to cells specified in the [Hcell](#) specification statement or using an hcell list. This statement may be preferable to the Hcell statement in nmDRC-H applications because of the flexibility of the Layout Cell List matching capabilities.

The cells in *list\_name* are not protected from user-specified hierarchy transformations which explicitly mention cell names, such as Expand Cell or Flatten Cell.

**nmLVS-H considerations** — Hcells apply during both connectivity extraction and netlist comparison, while the cells in Layout Preserve Cell List apply during connectivity extraction, but not during comparison. The Layout Preserve Cell List statement should be used only when it is absolutely essential that certain layout cells be preserved during connectivity extraction. Calibre nmLVS-H performs various hierarchy transformations for the purpose of optimizing run time and memory use. Exempting cells unnecessarily from these transformations is likely to reduce the effectiveness of these optimizations and to lead to an increase in run time and memory consumption. Hcells specified in LVS Preserve Cell List statements are subject to expansion by the [LVS Auto Expand Hcells](#) statement.

See also “[Suppressing Hierarchy Modification in LVS Connectivity Extraction](#)” in the *Calibre Verification User’s Manual*.

#### Example

These two statements declare all cells having names starting with nm\$\$ or pm\$\$ to be preserved cells. The preserved cells are protected from most optimizations of layout hierarchy normally performed during connectivity extraction.

```
LAYOUT CELL LIST pcells "nm$$*" "pm$$*"  
LAYOUT PRESERVE CELL LIST pcells
```

# Layout Primary

Specification statement

## LAYOUT PRIMARY *name*

Used only in Calibre.

### Parameters

- *name*

A required top-level cell or subcircuit name from the layout database.

The ***name*** parameter, which may be a singleton string variable, can contain one or more asterisk (\*) wildcard characters, where the (\*) character matches zero or more characters. When using \*, enclose the cell name in quotes; otherwise, a compilation error occurs because the asterisk is a reserved symbol.

### Description

Specifies a subcircuit name or cell name for GDSII, OASIS, Milkyway, OpenAccess, LEF/DEF, and SPICE layout systems for Calibre applications. Not used for the CNET, ASCII, or BINARY format layout systems. Can be specified once in your rule file.

This statement's function depends on the value of the [Layout System](#) specification statement as follows:

- Required for geometric layout inputs to identify the top-level cell name. Note that the ***name*** is case-sensitive. Only the top-level layout cell and cells below it in the layout hierarchy are processed.
- Optional for SPICE netlists, where it is used as follows:
  - If present, Layout Primary identifies the top-level subcircuit name. Calibre nmLVS evaluates the netlist starting with that subcircuit to the bottom of the hierarchy. The pins of the top-level subcircuit serve as design ports. Any subcircuit in the netlist can be specified as a top-level subcircuit.
  - When not present, nmLVS searches the netlist for element statements or subcircuit calls that are not part of any subcircuit. Those statements then constitute the top-level network.

The Layout Primary specification statement has special semantics if the wildcard character ( \* ) is present. These semantics allow the system to attempt a degree of auto-recognition of the top-cell in a geometric input layout database:

- If there is a literal match between a cell name and the Layout Primary ***name*** value, then that cell becomes the top cell.
- If there is no literal match and the Layout Primary ***name*** value contains one or more wildcards, then, instead of a fatal error, the system assembles a list of *candidate* top cells in order of their appearance in the input stream. A candidate top cell is defined as any unplaced (unreferenced) structure. The first such candidate whose name matches the

Layout Primary **name** value, according to the rules of cell name wildcard matching, is selected as the top cell (a warning is issued in this event).

- If there is still no match, then a fatal error occurs. Cell names in the input stream that contain “\*”, which are illegal in GDSII cell names, can literally match a Layout Primary **name** value also containing “\*”.

Use of the bare “\*” wildcard should be used with caution as this can cause unexpected results in some cases. Some Calibre applications, such as PERC, do not accept Layout Primary “\*”.

In Calibre Interactive, the Layout Primary cell is specified from the **Inputs > Layout** tab. See the [Calibre Interactive and Calibre RVE User’s Manual](#) for details.

See also [Layout Path](#), [Layout Primary2](#), and [Source Primary](#).

## Examples

```
LAYOUT PATH "./my_design.oasis"
LAYOUT SYSTEM OASIS
LAYOUT PRIMARY "ring"
```

## Layout Primary2

Specification statement

### LAYOUT PRIMARY2 *name*

Used only in Calibre.

#### Parameters

- *name*

A required top-level cell name or symbol of the second geometric input layout database.

The ***name*** parameter, which may be a singleton string variable, can contain one or more asterisk (\*) wildcard characters, where the \* character matches zero or more characters. When using \*, enclose the cell name in quotes; otherwise, a compilation error occurs because the asterisk is a reserved symbol.

#### Description

Specifies the second top-level cell name or symbol for GDSII and OASIS layout systems for dual-database Calibre applications. If you specify this statement in your rule file, then [Layout Path2](#), and [Layout System2](#) statements must also be specified.

Dual database comparison is described under “[Layout Versus Layout Comparison](#)” in the *Calibre Solutions for Physical Verification* manual. “[Dual Database Comparison](#)” in the *Calibre Verification User’s Manual*.

Note that the ***name*** is case-sensitive. Only the top-level layout cell and cells below it in the layout hierarchy are processed.

The Layout Primary2 specification statement has special semantics if the wildcard character (\*) is present. These semantics allow the system to attempt a degree of auto-recognition of the top-cell in a geometric input layout database.

- If there is a literal match between a cell name and the Layout Primary2 ***name*** value, then that cell becomes the top cell.
- If there is no literal match and the Layout Primary2 ***name*** value contains one or more wildcards, then, instead of a fatal error, the system assembles a list of *candidate* top-cells in order of their appearance in the input stream. A candidate top-cell is defined as any unplaced (unreferenced) structure. The first such candidate whose name matches the Layout Primary2 ***name*** value, according to the rules of cell name wildcard matching, is selected as the top-cell (a warning is issued in this event).
- If there is still no match, then a fatal error occurs. Cell names in the input stream that contain \*, which are illegal in GDSII cell names, can literally match a Layout Primary2 ***name*** value also containing \*.

See also [Layout Bump2](#).

### Examples

```
//SPECIFY FIRST DATABASE
LAYOUT PATH "/tmp/ring_new.gds"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM GDSII

//SPECIFY SECOND DATABASE
LAYOUT PATH2 "/tmp/ring.gds"
LAYOUT PRIMARY2 "TOP"
LAYOUT SYSTEM2 GDSII
LAYOUT BUMP2 200 // add 200 to layer numbers for this database

...
compare_1 { 1 XOR 201 }
compare_2 { 2 XOR 201 }
```

## Layout Process Box Record

Specification statement

### LAYOUT PROCESS BOX RECORD {NO | YES}

Used only in Calibre.

#### Parameters

- **NO**

Keyword that instructs the tool not to process BOX and BOXTYPE records. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to process BOX and BOXTYPE records.

#### Description

Specifies whether BOX and BOXTYPE records in GDSII input layout databases are processed. A DATATYPE record may appear in lieu of a BOXTYPE record and is handled as a BOXTYPE record in such cases.

BOX and BOXTYPE records are treated exactly like BOUNDARY and DATATYPE records, respectively, if you specify YES.

#### Examples

```
// Do not process BOX and BOXTYPE records in GDSII
LAYOUT PROCESS BOX RECORD NO
```

## Layout Process Node Record

Specification statement

### LAYOUT PROCESS NODE RECORD {NO | YES}

Used only in Calibre.

#### Parameters

- **NO**

Keyword that instructs the tool not to process NODE and NODETYPE records. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to process NODE and NODETYPE records.

#### Description

Specifies whether NODE and NODETYPE records in GDSII input layout databases are processed. A DATATYPE record may appear in lieu of a NODETYPE record and is handled as a NODETYPE record in such cases.

NODE and NODETYPE records are treated exactly like BOUNDARY and DATATYPE records, respectively, if you specify YES.

#### Examples

```
// Do not process NODE and NODETYPE records in GDSII
LAYOUT PROCESS NODE RECORD NO
```

# Layout Property Audit

Specification statement

**LAYOUT PROPERTY AUDIT** *prop\_name prop\_value [ prop\_value ... ]*  
 [ REPLACE | APPEND ]

Used only in Calibre nmDRC-H.

## Parameters

- *prop\_name*

A case-sensitive name or string constant that is a property name. If the parameter is a name preceded by the \$ character, then this represents an environment variable that is expected to resolve in the environment. If not, a compiler error results.

- *prop\_value*

A case-sensitive name or string constant that is a property value. If the parameter is a name preceded by the \$ character, then this represents an environment variable that is expected to resolve in the environment. If not, a compiler error results. More than one *prop\_value* may be specified per *prop\_name*.

- REPLACE

An optional keyword that specifies the *prop\_value* parameters are to replace the property values for the given *prop\_name* parameter from the database that is read in. This is the default behavior.

- APPEND

An optional keyword that specifies the *prop\_value* parameters are to be appended to the property values for the given *prop\_name* parameter from the database that is read in.

## Description

This statement allows property replacement, property insertion, or property modification of file-level properties in OASIS nmDRC results databases (GDSII databases are not modified).

The statement may be specified any number of times but only once for any given *prop\_name*.

The Layout Property Audit specification statement works as follows:

1. When the OASIS-type input layout database is read by Calibre nmDRC-H, all file-level properties  $P_1, \dots, P_n$  are saved. A file-level property is defined as a property that is attached to the START record of the (first) OASIS input layout database.

If the [Layout System](#) is GDSII or dual-database mode is in effect, then the set of properties  $P_i$  is defined to be empty and the algorithm proceeds.

2. Then for each OASIS-type nmDRC results database D written by nmDRC-H and for each Layout Property Audit specification statement L in the rule file, the following three steps are performed:
  - a. If REPLACE is specified in L either explicitly or by default, then the property in L is attached to the START record of D. The property's name is *prop\_name* and its values are the set of *prop\_value* parameters written as b-strings.
  - b. Otherwise, if none of the property names of the input properties  $P_i$  from step 1 coincide with the *prop\_value* of L, then the property in L is written as in step 2a.
  - c. Otherwise, for each input property  $P_i$  whose property name matches the *prop\_name* of L, a new property is written to D as in step 1, except that the property values are the set of *prop\_value* parameters of L appended in order to the existing property value list of  $P_i$ . Again, *prop\_value* is written as a b-string.

For details regarding OASIS results databases, see “[OASIS DRC Results Database Format](#)” in the *Calibre Verification User’s Manual*.

See also [DRC Results Database](#) and [DRC Check Map](#).

### Examples

The following statement inserts property “ver” with the string “1.0” into the OASIS results database:

```
DRC RESULTS DATABASE results.oas OASIS
DRC MAXIMUM RESULTS ALL
DRC KEEP EMPTY YES

LAYOUT PROPERTY AUDIT ver "1.0"
```

# Layout Property Text

Specification statement

## LAYOUT PROPERTY TEXT *attribute*

Used only in Calibre.

### Parameters

- *attribute*

A required positive integer, which specifies the property attribute that contains the property value to be used as text objects in an annotated GDSII layout.

### Description

The Layout Property Text specification statement allows GDSII properties to be treated as input layout database text. This statement is only used by Calibre applications when the input layout database format is GDSII, except Calibre xRC, which does not use annotated GDSII files. This statement can be specified any number of times.

This statement directs the application to treat properties attached to shapes, where *attribute* matches the value for GDSII PROPATTR, as input layout database text objects with the value of GDSII PROPVALUE as the name of the text object.

The layer of the text object is the same as that of the shape and the TEXTTYPE matches the DATATYPE of the shape. The location of the text object is an arbitrary point on the shape. The primitive layer numbers and texttypes for all text objects from a GDSII input layout database, including text objects specified by the Layout Property Text statement, are subject to mapping using the [Layer Map](#) TEXTTYPE specification statement prior to any other use or assignment.

Layers in annotated GDSII having the specified *attribute* number must also be identified as text layers in a [Text Layer](#) specification statement to participate in connectivity extraction.

See also [Layout Property Text OASIS](#).

### Examples

Suppose you have a polygon on layer 1 with attribute 100 = CLK, then you can do this to read the attribute as layout text:

```
LAYOUT PROPERTY TEXT 100    // recognize the property as text
TEXT LAYER 1                // use the text for connectivity extraction
```

# Layout Property Text OASIS

Specification statement

**LAYOUT PROPERTY TEXT OASIS** *name* [ *name* ... ]

Used only in Calibre.

## Parameters

- *name* [ *name* ... ]

A required property name, whose value is to be treated as input database text. You can specify multiple property names in one statement.

## Description

This statement allows OASIS properties to be treated as input layout database text. This statement is only used for OASIS layouts. This statement can be specified any number of times.

The layer of the text object is the same as that of the shape and the TEXTTYPE matches the DATATYPE of the shape. The location of the text object is an arbitrary point on the shape. The primitive layer numbers and texttypes for all text objects from a OASIS input layout database, including text objects specified by the Layout Property Text OASIS statement, are subject to mapping using the [Layer Map](#) TEXTTYPE specification statement prior to any other use or assignment.

The value of an OASIS property attached to an OASIS geometry is treated as an input layout database text object if the following conditions are met:

1. The name of the property matches (case-sensitive) a name in a Layout Property Text OASIS specification statement.
2. The property has exactly one value.
3. The type of the value is a-string, n-string, a-string reference, or n-string reference.

OASIS layers having the specified property *name* must also be identified as text layers in a [Text Layer](#) specification statement to participate in connectivity extraction.

See also [Layout Property Text](#).

## Examples

Suppose you have a polygon on layer 1 with propname = NET1, then you can do this to read the attribute as layout text:

```
LAYOUT PROPERTY TEXT OASIS propname // recognize the property as text
```

# Layout Rename Cell

Specification statement

**LAYOUT RENAME CELL** *source\_cell target\_cell*

Used only in Calibre.

## Parameters

- *source\_cell*  
A required name of a cell to be treated as if it were named *target\_cell*.
- *target\_cell*  
A required name to apply to the *source\_cell*.

## Description

Specifies cells to be renamed as the geometric input layout database is read by Calibre (see [Layout Path](#)). The *source\_cell* takes on the name of the *target\_cell*. This statement can be specified any number of times but only once for each *source\_cell*.

This statement can establish cell correspondences for dual-database comparison in hierarchical applications. Dual-database comparison is described in the [Calibre Verification User's Manual](#).

This statement does not rename the [Layout Primary](#) or [Layout Primary2 name](#) values. If you rename a top-level cell using Layout Rename Cell to a *target\_cell* name that is different from what appears in a Layout Primary or Layout Primary2 statement, this causes a fatal error. To rename a top-level cell, set the Layout Primary or Layout Primary2 statement to the *target\_cell* name.

See also [Layout Rename ICV](#).

## Examples

### Example 1

Rename cella to cellb

```
LAYOUT RENAME CELL cella cellb
```

### Example 2

Rename top cell

```
// LAYOUT PRIMARY TOP // commented out cell name
LAYOUT PRIMARY _TOP // new top cell name
LAYOUT RENAME CELL TOP _TOP
```

## Layout Rename ICV

Specification statement

### LAYOUT RENAME ICV {NO | YES}

Used only in Calibre.

#### Parameters

- **NO**

Keyword that instructs Calibre not to use unique identification of ICV cells across multiple files.

- **YES**

Keyword that instructs Calibre to use unique identification of ICV cells across multiple files. For example, in the case where Layout Path has multiple file parameters:

```
LAYOUT PATH <file1> <file2> ... <fileN>
```

Assume that  $file_i$  and  $i > 1$ , contains a cell named  $ICV\_*$ , where \* is any string of non-negative length. When this cell name matches a cell name in  $file_j$ , where  $j < i$ , then this cell is renamed to  $ICV\_*\$ICV<number>\$$ , where  $<number>$  is internally generated, and does not match a cell name in any  $file_j$  where  $j < i$ .

#### Description

That statement is used to enable unique identification of ICV cells across multiple files in the [Layout Path](#) or [Layout Path2](#) specification statement (but not across the two).

When pseudo cells are created by Calibre in the process of hierarchical injection, there is no control over their names, which are “ $ICV\_<number>$ ”. Hence, when running Calibre output back through Calibre (for example, in post-tapeout flows), multiple files in Layout Path can lead to cell name conflicts. You can resolve this by combining duplicate cell names ([Layout Input Exception Severity DUPLICATE\\_CELL](#)) but this is not the intent, since the two  $ICV\_<number>$  cells will only be the “same” by accident.

#### Examples

```
LAYOUT RENAME ICV NO
```

## Layout Rename Text

Specification statement

### LAYOUT RENAME TEXT

```
delimiter regular_expression delimiter replacement_pattern delimiter
[n | g] [e | b] [i | m] [Mc]
[delimiter regular_expression delimiter replacement_pattern delimiter
[n | g] [e | b] [i | m] [Mc]...]
[CELL LIST name]
[DATABASE] [RULES]
[BY LAYER layer [TO destination_layer]]
```

Used only in Calibre.

### Summary

Enables runtime editing of layout text values.

### Parameters

- ***delimiter***

A required single character (it can be any character except a space or newline) that acts as the delimiter for regular expression searches. You include this string three times in the statement, where specified in the syntax.

- ***regular\_expression***

A required regular expression that defines a pattern to be matched and replaced.

- ***replacement\_pattern***

A required string that replaces the pattern matched by the ***regular\_expression***.

- ***n | g***

An optional parameter that specifies which occurrences of ***regular\_expression*** are replaced with ***replacement\_pattern***. The possible choices are:

*n* — A positive integer that specifies to replace only the *n*th occurrence. The default behavior is *n*=1.

*g* — A literal that specifies to replace all occurrences.

- ***e | b***

An optional parameter that specifies which form of regular syntax is used. The possible choices are:

*e* — Specifies to use extended regular expression syntax. This is the default behavior.

*b* — Specifies to use basic regular expression syntax.

- **i | m**  
An optional parameter that specifies the case sensitivity when matching *regular\_expression*. The possible choices are:
  - i — Specifies to ignore case. This is the default behavior.
  - m — Specifies to match case.
- **Mc**  
An optional parameter, which includes a literal M followed by a character *c*, that specifies to use the character *c* instead of a backslash (\) as a metacharacter for creating expressions \0 - \9 in *replacement\_pattern*. The character *c* can be any character. There cannot be a space between M and *c*.
- **CELL LIST *name***  
An optional keyword and list name that restricts text values processed to certain cells. If this keyword is not specified, then the statement processes text values in all cells. The *name* parameter refers to a cell list specified with [Layout Cell List](#). This option only applies to hierarchical applications.
- **DATABASE**  
An optional keyword that specifies text objects in the GDSII or OASIS database are renamed. This is the default mode.
- **RULES**  
An optional keyword that specifies [Layout Text](#) or [Layout Text File](#) objects are renamed. This is the default mode.
- **BY LAYER *layer***  
An optional keyword set that specifies text objects are renamed only on the specified *layer*. The *layer* parameter may be an original layer number, layer name, or layer set. If it is a layer set, then the semantics are equivalent to the statement being repeated for each layer in the set.
- **TO *destination\_layer***  
An optional keyword set specified with BY LAYER, which maps *renamed* text objects from the BY LAYER *layer* parameter to the *destination\_layer*. The *destination\_layer* may or may not exist in the original database (GDSII or OASIS). The text moved to the *destination\_layer* is added to that layer. The *destination\_layer* may be an original layer name or number; it may not be a layer set.

### Description

Specifies text values to be edited or replaced. The statement edits the text value as it is being read. The resulting text value is used by Calibre applications. This statement can be specified any number of times.

This statement operates on all layout database text objects including connectivity extraction text, text from [Layout Text](#) or [Layout Text File](#) specification statements (controllable with the

RULES keyword), and text for **With Text** operations. It does not affect **Text** specification statements. In connectivity extraction, the layout netlist reflects renamed labels.

This statement searches each text value for an occurrence of ***regular\_expression***. By default, the first occurrence of ***regular\_expression*** is replaced by ***replacement\_pattern***. This default behavior can be modified with the use of the **varies** parameter option discussed under Parameters.

If you omit the ***regular\_expression*** and ***replacement\_pattern*** editing command section, then the regular expression “**./&/**” is assumed, which matches every text and does not rename it.

---

**Note**



Each set of parameters is considered one command. You can specify any number of commands for each Layout Rename Text specification statement.

---

Text values are edited in the following order:

1. Each Layout Rename Text statement is applied in the order in which it appears in the rule file.
2. Each command in a Layout Rename Text statement is applied to each text value. The resulting text values from the first command within the same statement are edited by the second command (if present), and so forth.
3. When at least one command in a Layout Rename Text statement changes a text value, the resulting text value is not edited by subsequent Layout Rename Text statements.

The **delimiter** parameter must be the first character of the command and must appear exactly three times. It can be any character except a space or newline. If you are trying to replace slash characters (/), use a different delimiter than /.

The ***regular\_expression*** parameter uses GNU implementation of regular expressions. Both basic and extended syntaxes are supported with the use of the appropriate flags as specified previously. The default is extended. The ***regular\_expression*** parameter can include regular subexpressions. Parentheses enclose regular subexpressions when using extended syntax. Subexpressions in basic syntax are enclosed as in the following example:

```
\(R\)\VDD
```

where R is the subexpression.

For more detailed information about regular expressions you may use the UNIX shell command:

```
man 5 regexp
```

The ***replacement\_pattern*** parameter replaces a string matched by the ***regular\_expression*** parameter, with the following exceptions:

- Ampersand (&): An & in ***replacement\_pattern*** is replaced with the string matching ***regular\_expression*** in the current command. You can precede & with a backslash (\) to suppress this behavior and literally place an & in ***replacement\_pattern***.

- Backslash followed by a single digit integer (\n): The characters \n, where n is a digit, are replaced by the text matched by the nth regular subexpression of the specified **regular\_expression** enclosed between “(” and “)”, or by “\(|“ and “\)|” when basic syntax is used. When nested, parenthesized subexpressions are present, n is determined by counting occurrences of “(” or “)”, starting from the left.
- Backslash (): If \ or & must appear as literals in the **replacement\_pattern**, they should be preceded by a single \ character: \\ becomes \ and \& becomes &. Note that because of special restrictions placed by the rule file compiler on the use of the \ character, it is *only* possible to replace the meta-character \ in **replacement\_pattern** by using the Mc parameter. This option has no effect on the **regular\_expression** part of the command. To avoid using \ in the **regular\_expression**, using extended regular expression syntax is recommended (this is the default).
- Empty string: You can leave **replacement\_pattern** empty to remove any occurrence of **regular\_expression** from the text value.

Special characters:

- The Calibre rule file compiler has special characters of its own, such as slash (/). Enclosing the commands in quotes is necessary if such characters are used. As shown in the following example:

```
LAYOUT RENAME TEXT "/A/B/g"
```

- The Calibre rule file compiler treats backslash () specially. It is used to create C-style characters, for example, \n becomes newline. This is done even inside quotation marks. To use a backslash in the Layout Rename Text statement, you must use a backslash as an escape character before the backslash used by the statement (that is, \\). Extended regular expression syntax and the Mc option significantly reduce the need for backslashes.
- You must precede the following characters with a backslash () if you use them as literals.

In extended syntax:

\	— everywhere
( ) [ ] { } * . ?	— in <b>regular_expression</b>
&	— in <b>replacement_pattern</b>

In basic syntax:

\	— everywhere
[ ] { } * .	— in <b>regular_expression</b>
&	— in <b>replacement_pattern</b>

You control which sources of layout text are ignored by using the DATABASE and RULES keywords. The default is DATABASE RULES, which means both text objects in the layout database and Layout Text statements in the rule file are ignored.

The keywords of this statement act as filters. For example, if DATABASE and BY LAYER are both specified, then only the text that is in the specified cell list, and that originates from the input database, and that is on the specified *layer* is renamed.

## BY LAYER Keyword

The BY LAYER keyword allows you to restrict renaming of text objects to a specific layer. When used with the TO keyword, text that appears on the BY LAYER layer, and is edited by a Layout Rename Text expression, is moved to the *destination\_layer*. Text that is not edited is not moved to the *destination\_layer*. To move all text on a given layer to another layer, with possible restrictions by cell or by text source, use a Layout Rename Text statement with no editing commands specified.

Note that the BY LAYER layer (source layer) for the text does not have to be declared in a [Text Layer](#) statement, a [Port Layer Text](#) statement, as a Device TEXT PROPERTY LAYER parameter, or any other kind of text layer. However, if a layer is not used as a text layer of some kind, then all text on this layer is ignored. Therefore, if the source layer is not used as a text layer of some kind, then all text that is not moved from this layer by Layout Rename Text statements is ignored. Similarly, if the *destination\_layer* is not used as a text layer, then all text that is moved to this layer is ignored.

Other filters that limit the scope of the Layout Rename Text specification statements apply when the *layer* parameter is specified. For example, if DATABASE and BY LAYER are both specified, then only the text that is in the specified cell list, and that originates from the input database, and that is on the specified *layer* is affected.

The BY LAYER and TO keywords do not affect the general rules for applying multiple Layout Rename Text statements (and multiple editing commands within one Layout Rename Text statement). All Layout Rename Text and [Layout Ignore Text](#) statements are applied to each text in the order they are written in the rule file. As soon as one of the statements performs its specified action, such as edits or ignores the text, subsequent statements are not applied to this text.

In particular, if a text is moved to a *destination\_layer* by a Layout Rename Text statement and there is a subsequent Layout Rename Text statement for that layer, the second statement is not applied. In each Layout Rename Text statement, all commands are applied to the text, in the order they are written in the statement, regardless of whether the earlier commands in the same statement resulted in any changes to the text or not.

## Examples

### Example 1

Consider these cases:

```
LAYOUT RENAME TEXT "/A/B/"
LAYOUT RENAME TEXT "/AA/MN/"
```

The second statement is never used because any text containing AA also contains A, so the first statement edits it and the editing stops there.

```
LAYOUT RENAME TEXT "/AA/MN/"  
LAYOUT RENAME TEXT "/A/B/"
```

Now all texts containing AA are edited by the first statement and only texts containing single A are edited by the second statement.

```
LAYOUT RENAME TEXT "/AA/MN/" "/BB/PQ/"  
LAYOUT RENAME TEXT "/A/B/"
```

Text AABB becomes MNPQ because all commands in one statement are executed in sequence. The second statement is not applied to this text because the first one changed it.

```
LAYOUT RENAME TEXT "/^VDD(.*)/VDD-1/M-"  
LAYOUT RENAME TEXT "/(.*)VDD(.*)/VDD-2_-1/M-"
```

The first statement catches all texts beginning with VDD and thus hides them from being edited by the second statement. The hyphen (-) character is being substituted for \ in the **replacement\_expression** due to the M-. Note that the first statement does not actually change any text, but any text that matches the command in it is considered edited and the second statement is not applied to it. Therefore, the text VDD:1 remains unchanged, while RVDD:1 becomes VDD:1\_R (compare with [Table 4-17](#)).

### Example 2

Assume you want to replace all occurrences of "(" with < and ")" with > for all text. This can be accomplished with the following:

```
LAYOUT RENAME TEXT "/\\(/</g" "/\\)/>/g"
```

**Table 4-17. Regular Expression Examples**

Regular expression	Input	Output	Notes
/A/B/	ABC	BBC	
/A/B/	AAA	BAA	only first match is replaced
/A/B/g	AAA	BBB	global replacement
/A/B/2	AAA	ABA	second match is replaced
/a/B/	ABC	BBC	case ignored by default
/a/B/m	Aa	Ab	input case matched
_V_P_	AVV	APV	any character can be delimiter
_V*_P_	AVV	APV	extended syntax is default

**Table 4-17. Regular Expression Examples (cont.)**

Regular expression	Input	Output	Notes
/(.*)VDD(.*)/VDD\2_\1	RVDD:1	VDD:1_R	see notes on backslash (\)
/(.*)VDD(.*)/VDD-2_-1/M-	RVDD:1	VDD:1_R	\ is replaced by - as metacharacter using M flag
/(.*)VDD(.*)/VDD-2_-1/M-	VDD:1	VDD:1_	(.*) matches empty substring
/^VDD.*/VCC/	VDD:1	VCC	^ searches from beginning of text
/A.*B/R/	ABAABBC	RC	longest leftmost string is matched when there is ambiguity
/AA.*//	AA.1		text removed
\n//g	ABC\n	ABC	remove all newlines
=/_=g	A/B/C	A_B_C	replace all / characters with _

**Example 3**

The following statements edit the same database text and rules file text differently: database text VDD is ignored, while rule file text VDD is replaced with PWR:

```
AYOUT RENAME TEXT "/^VDD$// " DATABASE
AYOUT RENAME TEXT "/^VDD$/PWR/" RULES
```

**Example 4**

This example demonstrates how the rules for multiple Layout Rename Text statements apply in presence of text source options.

The first statement replaces all text VDD by text PWR, regardless of the text source. If this statement edits a particular text, subsequent Layout Rename Text statements are not applied to this text. However, for text other than VDD, the next statement for the correct text source applies. In this case, the next applicable statement replaces VDD at the beginning of the text by RVDD if the text originates from the rule file, or by DVDD if the text originates from the database. Finally, the fourth statement applies to all text which contains VDD but not at the beginning of the text, and replaces VDD with POWER, again regardless of the source.

```
AYOUT RENAME TEXT "/^VDD$/PWR/" // 1
AYOUT RENAME TEXT "/^VDD/RVDD/" RULES // 2
AYOUT RENAME TEXT "/^VDD/DVDD/" DATABASE // 3
AYOUT RENAME TEXT "/VDD/POWER/" RULES DATABASE // 4
```

The following table illustrates the editing done by the above four statements:

**Table 4-18. Layout Rename Text Editing Examples**

Text	Source	Result	Comment
VDD	DATABASE	PWR	statement 1 applies
VDD	RULES	PWR	statement 1 applies
VDD1	DATABASE	DVDD1	statement 1 does not match the pattern; statement 2 does not match the source; statement 3 applies
VDD1	RULES	RVDD1	statement 2 applies
HVDD	DATABASE	HPOWER	statement 4 applies
HVDD	RULES	HPOWER	statement 4 applies
VSS	DATABASE	VSS	no statement applies

#### Example 5

The following statements replace text VCC by text PWR in all text on layer txt1:

```
LAYER txt1 1
TEXT LAYER txt1
LAYOUT RENAME TEXT "/^VCC$/PWR/" BY LAYER txt1
```

#### Example 6

The following statements delete all text containing GND on layer txt1, since replacing text by an empty string is equivalent to deleting it:

```
LAYER txt1 1
TEXT LAYER txt1
LAYOUT RENAME TEXT "/.*GND.*// " BY LAYER txt1
```

#### Example 7

The following statements move all text on layer txt1 in cells whose names start with “B” to layer txt2 in cells starting with “B.” Layer txt2 must be declared as an original layer using a [Layer](#) specification statement.

```
LAYER txt1 1
LAYER txt2 2
TEXT LAYER txt1 txt2
LAYOUT CELL LIST b "B*"
LAYOUT RENAME TEXT CELL LIST b BY LAYER txt1 TO txt2
```

The following statements are equivalent to the previous code because “./&/” matches every text but does not change it.

```
LAYER txt1 1
LAYER txt2 2
TEXT LAYER txt1 txt2
LAYOUT CELL LIST b "B*"
LAYOUT RENAME TEXT "./&/" CELL LIST b BY LAYER txt1 TO txt2
```

**Example 8**

The following statements move only text from layer txt1 to txt2 that contains letter “A”, in uppercase, without changing it. This applies to cells that start with “B.”

```
LAYER txt1 1
LAYER txt2 2
TEXT LAYER txt1 txt2
LAYOUT CELL LIST b "B*"
LAYOUT RENAME TEXT "/A/A/" CELL LIST b BY LAYER txt1 TO txt2
```

**Example 9**

The following statements move only text that begins with letter “X”, in uppercase, and remove the leading “X”:

```
LAYER txt1 1
LAYER txt2 2
TEXT LAYER txt1 txt2
LAYOUT RENAME TEXT "/^X///" BY LAYER txt1 TO txt2
```

**Example 10**

The following statements move all text on layer txt2 in cells whose name starts with “B” to layer txt1. Since layer txt2 is not a text layer, all text that remains on this layer is not used (assuming that layer txt2 is not declared elsewhere as some other kind of text layer).

```
LAYER txt1 1
LAYER txt2 2
TEXT LAYER txt1
LAYOUT CELL LIST b "B*"
LAYOUT RENAME TEXT CELL LIST b BY LAYER txt2 TO txt1
```

**Example 11**

See [Example 3](#) under Layout Ignore Text for a case when Layout Rename Text is also used.

# Layout System

Specification statement

**LAYOUT SYSTEM {GDS | GDS2 | GDSII} | OASIS | SPICE | LEFDEF | MILKYWAY | {OPENACCESS | OA} | CNET | ASCII | BINARY**

Used only in Calibre.

## Summary

Specifies the format of the layout design.

## Parameters

- **GDS | GDS2 | GDSII**

Keyword set that specifies that the layout is a GDSII or annotated GDSII stream file. All Calibre batch tools support version 6.0.

- **OASIS**

Keyword set that specifies that the layout is an OASIS file. All Calibre batch tools support version 1.0.

- **SPICE**

Keyword that specifies that the layout is a SPICE or HSPICE netlist. Used only for LVS netlist-to-netlist comparison.

- **LEFDEF**

Keyword that specifies that the layout is a set of LEF and DEF files, respectively. Calibre nmDRC, nmLVS, YieldAnalyzer, YieldEnhancer, xRC, and xL support version 5.x.

- **MILKYWAY**

Keyword that specifies the layout is a Milkyway database. Milkyway 2010.03 database schema 5.1 is supported.

- **OPENACCESS | OA**

Keyword that specifies the layout is an OpenAccess database. Calibre supports version 22.41p004. Not used by Calibre xRC or xL.

- **CNET**

Keyword that specifies that the layout is a LVS CNET database. Used for flat netlist-to-netlist comparison.

- **ASCII**

Keyword that specifies the layout is a file represented by the set of ASCII polygon files named icv\_data\_<number>. The tool searches for these files automatically. This is used only by flat applications, except for Calibre xRC or xL.

- **BINARY**

Keyword that specifies that the layout is a file represented by the set of binary polygon files named icv\_data\_<number>. The tool searches for these file automatically. This is used only by flat applications, except for Calibre xRC or xL.

### Description

Specifies the layout database type for Calibre applications. This statement must be specified once in your rule file. When specifying the Layout System, you also need to specify the [Layout Path](#) and [Layout Primary](#) statements, as follows:

**Table 4-19. Layout Input Specification Statement**

Layout System	Layout Path	Layout Primary
GDS   GDS2   GDSII	pathname of GDS file(s)	top-level cell name
OASIS	pathname of OASIS file(s)	top-level cell name
SPICE	pathname of the SPICE file	optional name of the top-level subcircuit
LEFDEF	pathnames of the LEF files or a directory of LEF files, followed by pathname of a DEF file (in xRC, it may be a directory of all DEF files)	top-level cell name
MILKYWAY	pathname of Milkyway library	top-level cell name; by default, the CEL view is read in
OPENACCESS   OA	name of the OpenAccess library specified in the lib.defs file	top-level cell name; by default the layout view is read in
CNET	pathname of CNET database	not used
ASCII	not used	not used
BINARY	not used	not used

For more details about supported databases, see the “[Layout Databases](#)” section of the *Calibre Verification User’s Manual*.

In Calibre Interactive, the Layout System is specified from the **Inputs > Layout** tab. See the [Calibre Interactive and Calibre RVE User’s Manual](#) for details.

See also [Layout System2](#) and [Source System](#).

### Treatment of Third-Party Layout Databases

Calibre tools do the following by default when reading LEF/DEF, Milkyway, and OpenAccess databases:

- Read the input cell names and use them for all internal processing.
- Preserve the text case of cell names.
- Map all input objects using their corresponding input layer numbers and a datatype record of 0.
- Read text objects.

For Milkyway and OpenAccess databases, the default order for opening cell views is the following:

- Milkyway: CEL, FRAM
- OpenAccess: layout, abstract

Calibre batch tools do not do the following by default when translating LEF/DEF, Milkyway, or OpenAccess databases:

- Abort when encountering an empty Pcell in an OpenAccess database.
- Use net name annotations from the input database.
- Use pin name annotations from the input database.
- Read fill shapes from Milkyway databases.
- Read properties from Milkyway or OpenAccess databases.

The MGC\_CALIBRE\_DB\_READ\_OPTIONS environment variable offers settings that modify the default read-in behavior for third-party databases. [Table 4-20](#) shows the variable settings and what they do. Select the hyperlinks to see detailed descriptions of the indicated arguments.

**Table 4-20. MGC\_CALIBRE\_DB\_READ\_OPTIONS Arguments**

Argument	Description
<a href="#">-abortOnEmptyPCell</a>	Causes the Calibre application to exit when an empty Pcell appears in an OpenAccess database. This option is only valid when -system OA is specified.
<a href="#">-annotateNets TEXT {ALL   TOP}</a>	Causes net names from the input database to be treated as text objects by Calibre tools. ALL creates text objects in all cells; TOP creates text objects at the top level only.
<a href="#">-annotatePins TEXT</a>	Causes pin names from the input database to be treated as text objects by Calibre tools.

**Table 4-20. MGC\_CALIBRE\_DB\_READ\_OPTIONS Arguments (cont.)**

Argument	Description
<b>-cellMap</b> <i>filename</i>	Enables mapping of cell names from the input database to names used internally by Calibre tools. The <i>filename</i> is a pathname of a text file that provides the mapping information. The MGC_CALIBRE_CELLMAP_FILE environment variable performs the same function. The MGC_CALIBRE_DB_OPTIONS variable setting takes precedence if both variables are specified. See <a href="#">Example 3</a> later in this section.
<b>-layerMap</b> <i>filename</i>	Enables mapping of layer names from the input database to names used internally by Calibre tools. The <i>filename</i> is a pathname of a text file that provides the mapping information. The MGC_CALIBRE_LAYERMAP_FILE environment variable performs the same function. The MGC_CALIBRE_DB_OPTIONS variable setting takes precedence if both variables are specified. See <a href="#">Example 3</a> later in this section.
<b>-mapInfo</b> <i>filename</i>	Causes a text file to be written to the specified <i>filename</i> . The file contains the current cell and layer mapping information from either the default settings, or the -cellMap or -layerMap arguments.
<b>-noText</b>	Causes text objects not to be read from the input database.
<b>-outputBlockages</b>	Causes blockages to be read from LEF/DEF and OpenAccess databases, which they are not by default.
<b>-outputFills</b>	Causes fill shapes to be read from the Milkyway database.
<b>-viewList</b> <i>view1 view2 ...</i>	Causes the cell views to be opened in the specified order. Applies only to Milkyway and OpenAccess databases. The default order for opening views is described previously in this section.

Multiple MGC\_CALIBRE\_DB\_READ\_OPTIONS environment variable parameters may appear in a space-delimited list, as follows.

Bourne shell:

```
MGC_CALIBRE_DB_READ_OPTIONS="-option1 arg -option2 arg ..."
export MGC_CALIBRE_DB_READ_OPTIONS
```

C shell:

```
setenv MGC_CALIBRE_DB_READ_OPTIONS "-option1 arg -option2 arg ..."
```

For example, in the C shell, you can set the following variable to enable both top-level net and pin name annotation with text objects:

```
setenv MGC_CALIBRE_DB_READ_OPTIONS "-annotateNets TEXT TOP -annotatePins TEXT"
```

The MGC\_CALIBRE\_DB\_READ\_OPTIONS variable applies to the [Layout System2](#) design if it is OpenAccess.

You can perform a fully-customized translation of your LEF/DEF, Milkyway, or OpenAccess databases to either GDS or OASIS by using the FDI utilities. These utilities are discussed under “[Translate Third-Party Layout Databases](#)” in the *Calibre Verification User’s Manual*.

For information regarding how Calibre Interactive handles third-party layout databases, see “[Specifying Database Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## Examples

### Example 1

Specify a GDSII layout database

```
LAYOUT SYSTEM GDS
LAYOUT PATH "/home/designs/chip.gds"
LAYOUT PRIMARY "TOP"
```

### Example 2

Specify a SPICE-to-SPICE LVS comparison

```
LAYOUT SYSTEM SPICE
LAYOUT PATH "layout.spi"
LAYOUT PRIMARY memory_block

SOURCE SYSTEM SPICE
SOURCE PATH "/home/spice_sources/mem.spi"
SOURCE PRIMARY memory_block
```

### Example 3

This example describes using OpenAccess as the Layout System. By default, input layers are mapped using the input layer numbers and a datatype record of 0. Assume you want to override this behavior and you want to map specific input layers to a user-specified list of output layers and datatypes. You can do this by writing the following mapping to a text file. Create a file called layermap.txt, as follows:

```
# input_layer_name    purpose    output_layer    datatype
met1          drawing      11            1
met2          drawing      12            1
met3          drawing      13            1
```

This file maps the indicated input layers to the output layer numbers and datatypes.

By default, the cell names that appear in the OpenAccess database are used on input. Assume you want to override this behavior and you want to rename the cells. Create a file called cellmap.txt, as follows:

```
# library    input_name   view_name   output_name
DesLib      nand2.1     layout       nand2x1
DesLib      nand2.2     layout       nand2x2
```

This file maps the indicated input cell name from the specified design library and view to the output cell name.

You can then set the following environment variable to perform your layer mapping and cell mapping. C shell is assumed.

```
setenv MGC_CALIBRE_DB_READ_OPTIONS "-layerMap layermap.txt -cellMap cellmap.txt"
```

The following can be specified in the rule file to read the OpenAccess database.

```
LAYOUT SYSTEM OA
LAYOUT PATH "DesLib" //library defined in the lib.defs file
LAYOUT PRIMARY top
```

The lib.defs file would contain something like the following entry:

```
DEFINE DesLib /home/libraries/DesLib
```

The environment variable specified previously is read automatically when the Calibre job is executed and the input database is processed accordingly.

## Layout System2

Specification statement

**LAYOUT SYSTEM2 {{GDS | GDS2 | GDSII} | OASIS | LEFDEF | {OA | OPENACCESS}}}**

Used only in Calibre.

### Parameters

- **GDS | GDS2 | GDSII**  
Keyword that specifies that the layout is a GDSII stream file.
- **OASIS**  
Keyword that specifies that the layout is OASIS format.
- **LEFDEF**  
Keyword that specifies that the layout is LEF/DEF format.
- **OA | OPENACCESS**  
Keyword that specifies the layout is OpenAccess format.

### Description

Specifies the second layout database type for dual-database Calibre applications. If you specify Layout System2, then [Layout Path2](#) and [Layout Primary2](#) specification statements are also required. This statement may be specified once in your rule file.

Comparison of differing layout formats is supported. Hence the [Layout System](#) may be different from Layout System2. A Calibre nmDRC-H license is required for heterogeneous comparison runs.

See “[GDSII Layout Format](#)” in the *Calibre Verification User’s Manual* for details about how Calibre treats GDSII format. See [Calibre for the Open Artwork System Interchange Standard](#) for more information about OASIS use in the Calibre toolset.

If LEFDEF or OPENACCESS is specified, and if the MGC\_CALIBRE\_LAYERMAP\_FILE or the MGC\_CALIBRE\_DB\_READ\_OPTIONS variable is set, those variable settings apply to the Layout Path2 design also. See “[Treatment of Third-Party Layout Databases](#)” on page 798 for details.

Dual database comparison is described under “[Layout Versus Layout Comparison](#)” in the *Calibre Solutions for Physical Verification* manual. “[Dual Database Comparison](#)” in the *Calibre Verification User’s Manual*.

See also [Layout Bump2](#).

## Examples

### Example 1

```
//SPECIFY FIRST DATABASE
LAYOUT PATH "/tmp/ring_new.gds"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM GDSII

//SPECIFY SECOND DATABASE
LAYOUT PATH2 "/tmp/ring.gds"
LAYOUT PRIMARY2 "TOP"
LAYOUT SYSTEM2 GDSII
LAYOUT BUMP2 200 // add 200 to layer numbers for this database

...
compare_1 { 1 XOR 201 }
compare_2 { 2 XOR 201 }
```

### Example 2

Assume you have this in the rule file:

```
//SPECIFY FIRST DATABASE
LAYOUT PATH "/tmp/design1.gds"
LAYOUT PRIMARY "TOP"
LAYOUT SYSTEM OPENACCESS

//SPECIFY SECOND DATABASE
LAYOUT PATH2 "/tmp/design2.gds"
LAYOUT PRIMARY2 "TOP"
LAYOUT SYSTEM2 OPENACCESS
LAYOUT BUMP2 200 // add 200 to layer numbers for this database
```

And suppose you have the MGC\_CALIBRE\_LAYERMAP\_FILE set to a file that has the following contents:

```
metal1 drawing 11 0
m1      drawing  11 0
```

Then the specified layer mapping applies to both input designs because they are both OpenAccess.

# Layout Text

Specification statement

**LAYOUT TEXT** *name* *x* *y* *layer* [*texttype*] [*cell\_name*]

Used only in Calibre.

## Parameters

- ***name***  
A required name of a layout database text object.
- ***x* *y***  
A pair of required floating-point user unit coordinates in the cell space of *cell\_name*.
- ***layer***  
A required layer number or the name of an original layer or layer set. If *layer* is a simple layer name, that is equivalent to use of its layer number in the statement. If it is a layer set name, then that is equivalent to the statement being repeated for each element of the layer set.
- ***texttype***  
An optional non-negative integer that indicates a text object's text type. The value of *texttype* defaults to 0 when you do not include this parameter in the statement.
- ***cell\_name***  
An optional name of a cell in which the text object is to be placed. If not specified, the default is the top-level cell. Asterisks (\*) in this parameter are treated as literal characters and not wildcards.

## Description

Allows free-standing cell-based text to be specified directly in the rule file. The text object *name* behaves exactly as if it were in the layout database in *cell\_name* at coordinates *x* *y* in the cell coordinate space on *layer*. See the section “Use of Text in Calibre Applications” in the [Calibre Verification User’s Manual](#).

This statement can be specified any number of times, and it can only be used when the [Layout System](#) is geometric. The layout reader issues a warning for and then ignores each Layout Text object whose *cell\_name* is not present in the input database. The *layer* parameter should be a [Text Layer](#) argument if the layer is used to name nets for connectivity extraction.

Layout Text is in particular useful in hierarchical nmLVS for specifying local net names in cells. Additionally, all verification applications use Layout Text objects as top-level text with the appropriate [Text Depth](#) setting.

If you have many Layout Text objects to specify, this can slow down rule file compilation. Use the [Layout Text File](#) statement instead.

Object coordinates in user units generated using this statement are not affected by changes in the [Precision](#) setting. However, coordinates in database units do follow the Precision setting. For example, if you specify (1, 1) as the coordinates of an object and the Precision is 1000, the object is at (1000, 1000) in database units. If you change the Precision to 10000, the user coordinates remain at (1, 1) but the database coordinates become (10000, 10000).

Text objects generated by this statement are affected by [Layout Magnify](#) with an explicit *value*; the **AUTO** keyword has no effect, however. The [Layout Path](#)[2] MAG keyword has no effect on Layout Text.

All text objects from a GDSII or OASIS input layout database (including those specified by Layout Text specification statements) have their primitive layer numbers and texttypes subject to mapping through [Layer Map](#) TEXTTYPE specification statements prior to any other use or assignment.

Differences between text objects in [Text](#) and Layout Text specification statements are as follows:

- Text statement objects are always in top-level coordinates and have no cell association. Layout Text statements have a cell association and have cell-based coordinates.
- Text statement objects may edit existing database text within Calibre, whereas Layout Text objects cannot.
- Layout Text objects may be used for connectivity extraction, used in (Not) [With Text](#) operations, used as model name text in [Device](#) operations, used as ports, or used as [DRC Map Text](#) objects. Text statement objects are used only for connectivity extraction.
- Layout Text objects used for connectivity extraction observe [Text Layer](#) and [Text Depth](#) requirements (as with any database text used for connectivity extraction) whereas Text statement objects do not.
- Layout Text objects may have texttypes and obey [Layer Map](#) statements. Text statement objects do not have these features.
- Layout Text objects are used only for geometric layout databases. Text statement objects are used for any layout database type.

See also [Attach](#), [Label Order](#), [Layout Rename Text](#), [Port Depth](#), and [Port Layer Text](#).

## Examples

```
//Place layout text "vdd" in cell muxblock on layer 17
//at cell coordinates -3.7 5.9
TEXT LAYER 17
LAYOUT TEXT vdd -3.7 5.9 17 muxblock
```

# Layout Text File

Specification statement

## LAYOUT TEXT FILE *filename*

Used only in Calibre.

### Parameters

- *filename*

A required pathname of a file containing layout text statements.

### Description

Allows free-standing cell-based text to be specified in the *filename*.

This statement should be used instead of [Layout Text](#) statements when you have many Layout Text objects to specify. Rather than specifying them in the rule file directly with Layout Text, which can greatly slow down rule file compilation, it is more efficient to use a Layout Text File statement. The *filename* is processed quickly because the allowed syntax is limited in comparison to a standard rule file.

The format of the layout text file is as follows:

- Empty lines and whitespace are ignored, just as in regular SVRF.
- Lines beginning with // are ignored and can be used for comments.
- No statements other than layout text specifications can appear in the file, including Layout Text File statements (that is, nesting of Layout Text Statements is forbidden).
- Layout text specifications in the *filename* follow the same syntax as Layout Text statements, although the words “Layout Text” may be omitted. Here is the syntax:

[LAYOUT TEXT] *name* *x* *y* *layer* [*texttype*] [*cell\_name*]

The text object *name* behaves exactly as if it were in the layout database in *cell\_name* at coordinates *x* *y* in the cell coordinate space (user units) on *layer*. See the section “Use of Text in Calibre Applications” in the [Calibre Verification User’s Manual](#).

This statement can be specified any number of times, and it can only be used when the [Layout System](#) is geometric. The layout reader issues a warning for and then ignores each layout text file object whose *cell\_name* is not present in the input database. The *layer* parameter should be a [Text Layer](#) argument if the layer is used to name nets for connectivity extraction.

File text is particularly useful in hierarchical nmLVS for specifying local net names in cells. Additionally, all verification applications use layout text file objects as top-level text with the appropriate [Text Depth](#) setting.

Object coordinates in user units generated using a layout text file are not affected by changes in the [Precision](#) setting. However, coordinates in database units do follow the Precision setting. For example, if you specify (1, 1) as the coordinates of an object and the Precision is 1000, the

object is at (1000, 1000) in database units. If you change the Precision to 10000, the user coordinates remain at (1, 1) but the database coordinates become (10000, 10000).

For the purpose of magnification, layout text file objects generated are treated as if they exist in the first (or only) file of the [Layout Path](#) statement. Whatever magnification applies to that design applies to Layout Text objects.

All text objects from a GDSII or OASIS input layout database (including those specified by Layout Text File specification statements) have their primitive layer numbers and texttypes subject to mapping through [Layer Map](#) TEXTTYPE specification statements prior to any other use or assignment.

Differences between text objects generated by [Text](#) and Layout Text File specification statements are as follows:

- Text statement objects are always in top-level coordinates and have no cell association. Layout text file objects have a cell association and have cell-based coordinates.
- Text statement objects may edit existing database text within Calibre, whereas layout text file objects cannot.
- Layout text file objects may be used for connectivity extraction, used in (Not) [With Text](#) operations, used as model name text in [Device](#) operations, used as ports, or used as [DRC Map Text](#) objects. Text statement objects are used only for connectivity extraction.
- Layout text file objects used for connectivity extraction observe [Text Layer](#) and [Text Depth](#) requirements (as with any database text used for connectivity extraction) whereas Text statement objects do not.
- Layout text file objects may have texttypes and obey [Layer Map](#) statements. Text statement objects do not have these features.
- Layout text file objects are used only for geometric layout databases. Text statement objects are used for any layout database type.

See also [Attach](#), [Label Order](#), [Layout Rename Text](#), [Port Depth](#), and [Port Layer Text](#).

## Examples

```
TEXT LAYER txt
// Use a textng file
LAYOUT TEXT FILE layout_text
```

## Layout Top Layer

Specification statement

**LAYOUT TOP LAYER** *layer* [*layer* ...]

Used only in hierarchical Calibre.

### Parameters

- *layer*

A required name of an original (drawn) layer. You can specify *layer* any number of times in one statement.

### Description

---

#### Note



Use of [Layout Base Layer](#) or Layout Top Layer is strongly recommended for all hierarchical applications. Layout Base Layer is usually preferable from a rule file maintenance standpoint.

---

Specifies top-level layers for performance tuning of hierarchical applications. The hierarchical database constructor creates an internal model of the design hierarchy that is optimal for runtime performance. The primary optimization that this statement provides is the extremely important ability for Calibre to distinguish cells whose primary function is routing from cells that comprise the *device plane* of the design. The former are often called *top-layer cells*.

Layout Top Layer enumerates a set of original (drawn) layer names that are considered to be top-level layers. These layers are generally distinct from device-forming layers. Cells that contain only top layers as original data (including through the sub-hierarchy of the cell) are identified as top-layer cells and are subjected to a host of internal optimizations.

Typically, top layers contain data configurations that are unique (not replicated anywhere else on a mask). Recommended layers to include are all metal and via (not contact) layers from first layer metal to the top; anomalies like solder bumps, fuses, and pads; and special purpose layers such as markers, artificial cell boundaries, and text. Device-forming layers (poly, diffusion, implant, contact, and so forth) should not be included.

Layers specified in this statement must be required by the Calibre run (that is, they must be read in order to produce results output), otherwise top-layer cells may not be recognized properly. This restriction does not apply to Layout Base Layer.

In some cases, a metal layer can be used in a device-level (seed) layer within cells, but not for interconnect routing between cells. In these cases, the metal layer used in the device-level layer should not be declared in a Layout Top Layer statement (consider using Layout Base Layer instead of Layout Top Layer in this case, and declaring the metal layer as a base layer). If a metal layer is used both in the device-level layer and to connect cells, specify the metal layer in a Layout Top Layer statement.

This statement should be used when the design incorporates high-level power rails or bumps as placements instead of geometry. Typically, these placements cover the entire design, which can cause the system heuristics problems in distinguishing such a design from a gate array.

This statement is a counterpart to [Layout Base Layer](#), which is an easier statement to maintain. If Layout Base Layer and Layout Top Layer appear in the same rule file, Layout Top Layer is ignored. Specifying Layout Top Layer invokes the same optimization heuristics as Layout Base Layer for layers that are considered top layers. A layer L is a top layer if either of the following is true:

- A Layout Base Layer statement is specified, and L is not specified in a Layout Base Layer Statement.
- No Layout Base Layer statement is specified and is in a Layout Top Layer statement.

You can specify this statement any number of times and use it only with hierarchical Calibre applications. In nmLVS-H, specified top layer cells are expanded by one level.

**Cell pushdown heuristics optimization** — If either Layout Base Layer or Layout Top Layer is properly specified, then only top-layer cell placements are candidates for pushdown and non-top-layer cell placements are never pushed down. Top-layer cell placements are placements of cells that do not contain any layers other than those that are considered top-level, as defined previously. Cell pushdown is an optimization in the hierarchical database constructor that, under certain conditions, pushes cell placements down into underlying cells.

If neither Layout Base Layer nor Layout Top Layer are specified, then all layers are treated as base layers, and the usual criteria are used to select candidates for pushdown (specifically, very small cell placements are candidates for pushdown). When Layout Base Layer or Layout Top Layer is properly specified, the hierarchical database constructor in Calibre nmLVS-H does not push cells that contain devices down into other cells. This prevents undesired pushdown of devices into hcells.

## Examples

```
// specify the interconnect routing layers and other non-device layers
LAYOUT TOP LAYER m6 v5 m5 v4 m4 v3 m3 v2 m2 v1 m1
LAYOUT TOP LAYER bump pad
```

## Layout Use Database Precision

Specification statement

### LAYOUT USE DATABASE PRECISION {NO | YES}

Used only in Calibre.

#### Parameters

- **NO**

Required keyword indicating the rule file **Precision** is used. This is the default for most Calibre applications if you do not specify this statement.

- **YES**

Required keyword that indicates the input database precision overrides the rule file precision. For Calibre xRC and Calibre xL, this is the default if you do not specify this statement.

#### Description

Allows the precision of the input layout database for Calibre applications to override the Precision of the rule file (or its default). This statement may be specified at most once.

The Calibre rule file compilation module works as follows when you specify YES:

1. Calibre compiles the rule file as its first task, as usual. The rule file Precision, or its default of 1000, is used. However, this is only a partial compilation. Litho, Fracture, and MDP setup files are not syntax-checked, and the compilation error **NUM8** for the **Size** operation is not enforced. Any other compilation error aborts Calibre, as usual.
2. If the **Layout System** is geometric, then the first file name parameter of the **Layout Path** specification statement, if present, is opened. Enough of the corresponding input layout database is read to ascertain its precision. Any open or read error aborts Calibre.
3. The rule file is recompiled in its entirety as if the precision value garnered in Step 2 were the value of the Precision specification statement. If no precision value was garnered in Step 2, then the rule file is simply compiled again. Any compilation error aborts Calibre, as usual.

See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

#### Examples

```
// use the rule file PRECISION  
LAYOUT USE DATABASE PRECISION NO
```

## Layout Windel

Specification statement

**LAYOUT WINDEL *x1 y1 x2 y2 [xN yN ...]***

Used only in Calibre.

### Parameters

- ***x1 y1 x2 y2***

A set of required floating-point numbers that specifies the coordinates of vertices of a polygon in user units. Exactly two pairs of coordinates are interpreted as the lower-left and upper-right corners (they must be specified in that order) of a rectangle having sides parallel, respectively, to the coordinate axes. You must specify at least two coordinate pairs. You can specify *x* and *y* values in a [Variable](#) statement.

### Description

Specifies a single polygon window that defines the *exclusion* of input shapes and text to be used in subsequent rule checks. Database objects totally outside or intersecting the specified window are processed. Objects that are inside coincident with the specified window are not processed.

This statement can be specified any number of times. A specified window must be a simple, orientable polygon of any shape.

When [Layout Window Clip](#) YES is used in conjunction with this statement, only those portions of intersecting database objects that fall outside the polygon window are processed. Using Layout Window Clip NO with this statement results in the same output as specifying Layout Windel alone (shapes that overlap the boundary of the Layout Windel region are processed whole).

If Layout Windel is specified with a [Layout Window](#) statement, the resulting window is defined as the Boolean NOT of Layout Window with Layout Windel. Database objects within or intersecting the resultant window are processed.

Filtering for nmDRC-H is slightly more complicated in that, starting from the top level down, cell placements having extents that cross the data *inclusion* region for the run are flattened one level. The process is recursive until no cell placements cross the inclusion region. Shapes, cell placements, and text are then filtered as described previously. The database extent is recomputed to account for any window filtering.

Layout Windel does not observe layout magnification by the [Layout Magnify](#) statement or the MAG keyword of statements that support MAG.

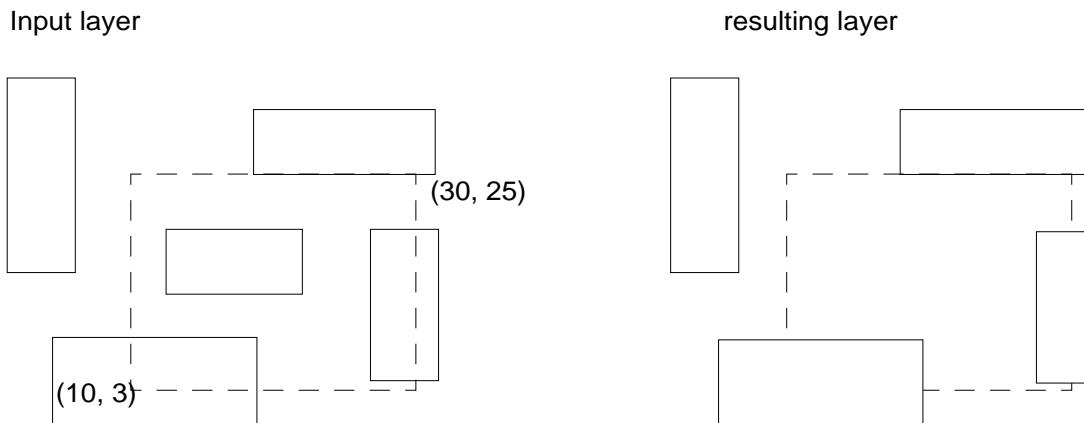
See also [Layout Windel Cell](#), [Layout Windel Layer](#).

**Examples**

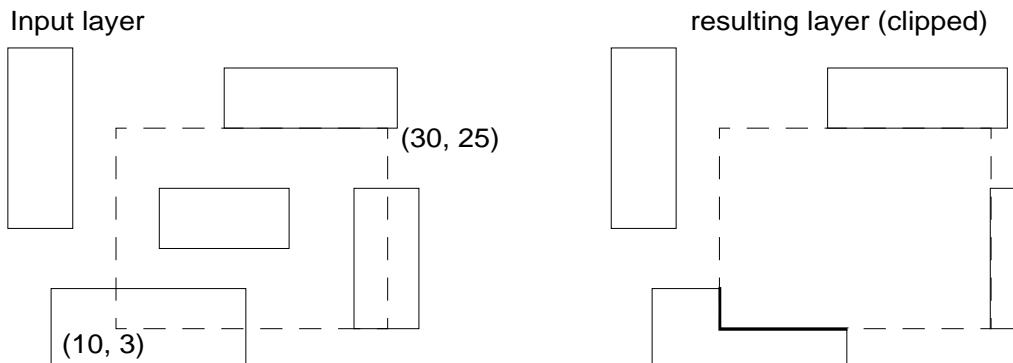
Figure 4-146 shows that the three database objects completely or partially outside the exclusion window defined by Layout Windel are used for processing. Figure 4-147 shows the same shapes and Layout Windel statement with the addition of the Layout Window Clip YES.

**Figure 4-146. Layout Windel**

LAYOUT WINDEL 10 3 30 25

**Figure 4-147. Layout Windel with Layout Window Clip YES**

LAYOUT WINDEL 10 3 30 25  
LAYOUT WINDOW CLIP YES



# Layout Windel Cell

Specification statement

**LAYOUT WINDEL CELL** *cell\_name* [*cell\_name* ...] [ORIGINAL [OCCUPIED]]

Used only in Calibre.

## Parameters

- *cell\_name*

Required name of a cell. Any number of cell names may be specified. Each *cell\_name* parameter may contain one or more asterisk (\*) characters; the \* character is a wildcard that matches zero or more characters. When using the \* character, be sure to enclose the cell name in quotation marks (" ")—otherwise a compilation error occurs because \* is a reserved symbol. Specifying \* alone includes the top cell. String variables can also be used for cell names (see [Variable](#) for details).

- ORIGINAL

Optional keyword that specifies all objects in the input layout database are used to compute the specified cell extent(s).

- OCCUPIED

Optional keyword used with ORIGINAL that specifies only the cells containing geometry required in the Calibre run (including anywhere in their subhierarchy) have their original extents returned; all other cells are ignored.

## Description

Specifies to exclude shapes and text objects that occur in windows formed for each *cell\_name* during a Calibre run. This statement functions essentially like [Layout Windel](#), but rather than providing coordinates, you provide cell names. For each cell C, the extents of all placements of C in the input layout database from the flat viewpoint are scanned. The coordinates of each extent are then used in an implicit Layout Windel statement.

Calibre warns for each *cell\_name* argument that is not located in the input layout database.

If ORIGINAL is specified, then all objects in the input layout database are used to compute the specified cell extent(s), not just those objects actually required by the flow. If OCCUPIED is additionally specified, then all objects in the input layout database are used to compute the specified cell extent(s), if there is at least one object required by the flow within the cell or its subhierarchy; however, if a specified cell does not actually contain objects that are required by the flow (either in the cell or its sub-hierarchy), then the extent returned by this operation for that cell is empty (that is, as if ORIGINAL were not specified).

By default, the cell extents used by these statements are the extents of only those objects (text and geometry) actually required by the run. Calibre applications completely ignore objects in the input layout database that are not actually required by the flow. For example, if the input layout database contains objects on layer 53, but layer 53 is nowhere in a rule check or is not required for connectivity in the current run, then this layer is ignored as the input layout

## **Layout Windel Cell**

---

database is processed. Being ignored implies that objects on layer 53 are not used to compute the extents of individual cells.

See also [Layout Windel Layer](#), [Layout Window Cell](#), [Layout Window Clip](#), and [Exclude Cell](#).

### **Examples**

#### **Example 1**

All extents of `dff_*` and memblock instances become exclusion windows; use all objects in input database for computing extents.

```
Layout Windel Cell "dff_*" memblock ORIGINAL
```

#### **Example 2**

A diagram of Layout Windel Cell behavior appears in [Figure 4-151](#) on page 819.

# Layout Windel Layer

Specification statement

**LAYOUT WINDEL LAYER** *layer* [*layer* ...]

Used only in Calibre.

## Parameters

- *layer*

Required original layer or layer set. Any number of layers may be specified.

## Description

Specifies to exclude polygons that occur in windows formed for each *layer* during a Calibre run. This statement behaves essentially like [Layout Windel](#), but rather than providing coordinates, you provide layer names. For each layer L, all shapes on L are scanned from the input layout database and transformed to top-level cell space. The coordinates of the vertices of each polygon on L are then used to form an implicit [Layout Windel](#) statement.

This statement may be specified any number of times.

See also [Layout Windel Cell](#), [Layout Window Layer](#), and [Layout Window Clip](#).

## Examples

### Example 1

Make all nwell polygons into exclusion windows, including polygons that overlap.

```
LAYOUT WINDEL LAYER nwell
```

### Example 2

The following example yields no DRC results inside the specified layer:

```
LAYOUT WINDOW LAYER layerA  
LAYOUT WINDEL LAYER layerA
```

### Example 3

A diagram of Layout Windel Layer behavior appears in [Figure 4-153](#) on page 822.

## Layout Window

Specification statement

**LAYOUT WINDOW  $x1\ y1\ x2\ y2\ [xN\ yN\ ...]$**

Used only in Calibre.

### Parameters

- **$x1\ y1\ x2\ y2$**

A set of required floating-point numbers that specifies the coordinates of vertices of a polygon in user units. Exactly two pairs of coordinates are interpreted as the lower-left and upper-right corners (they must be specified in that order) of a rectangle having sides parallel, respectively, to the coordinate axes. You must specify at least two coordinate pairs. You can specify **x** and **y** values in a [Variable](#) statement.

### Description

Specifies a single window that defines the *inclusion* of input shapes and text to be used in rule checks. Database objects totally inside or intersecting the polygon window are processed. Objects that are outside coincident (that is, they touch the window only at an edge) are not processed.

This statement can be specified any number of times. A specified window must be a simple, orientable polygon of any shape.

When [Layout Window Clip YES](#) is used in conjunction with this statement, only those portions of intersecting database objects that fall inside the inclusion window are processed. Using [Layout Window Clip NO](#) with this statement results in the same output as specifying Layout Window alone (shapes crossing the boundary of the window are processed whole).

If Layout Window is specified with a [Layout Windel](#) statement, the resulting window is defined as the Boolean NOT of Layout Window with Layout Windel. Database objects within or intersecting the resultant window are processed.

Filtering for nmDRC-H is slightly more complicated in that, starting from the top level down, cell placements having extents that cross the data *inclusion* region for the run are flattened one level. The process is recursive until no cell placements cross the inclusion region. Shapes, cell placements, and text are then filtered as described previously. The database extent is recomputed to account for any window filtering.

Layout Window does not observe layout magnification by the [Layout Magnify](#) statement or the MAG keyword of statements that support MAG.

See also [Layout Window Cell](#) and [Layout Window Layer](#).

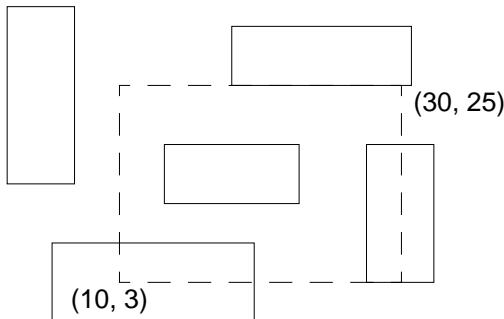
## Examples

Figure 4-148 shows that the three database objects completely or partially inside the polygon window defined by Layout Window are used for processing. Figure 4-149 shows the same shapes and Layout Window statement with the addition of Layout Window Clip YES.

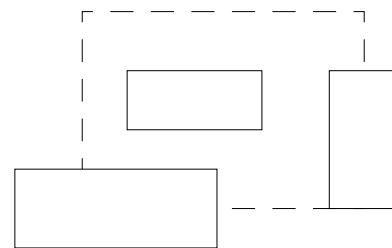
**Figure 4-148. Layout Window**

LAYOUT WINDOW 10 3 30 25

Input layer



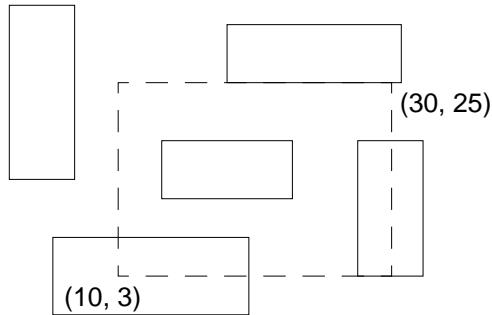
resulting layer



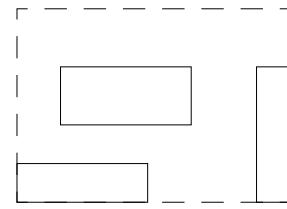
**Figure 4-149. Layout Window with Layout Window Clip YES**

LAYOUT WINDOW 10 3 30 25  
LAYOUT WINDOW CLIP YES

Input layer



resulting layer (clipped)



## Layout Window Cell

Specification statement

**LAYOUT WINDOW CELL** *cell\_name* [*cell\_name* ...] [ORIGINAL [OCCUPIED]]

Used only in Calibre.

### Parameters

- *cell\_name*  
Required name of a cell. Any number of cell names may be specified. Each *cell\_name* parameter may contain one or more asterisk (\*) characters; the \* character is a wildcard that matches zero or more characters. When using the \* character, be sure to enclose the cell name in quotation marks ("")—otherwise a compilation error occurs because \* is a reserved symbol. Specifying \* alone includes the top cell. String variables can also be used for cell names (see [Variable](#) for details).
- ORIGINAL  
Optional keyword that specifies all objects in the input layout database are used to compute the specified cell extent(s).
- OCCUPIED  
Optional keyword used with ORIGINAL that specifies only the cells containing geometry required in the Calibre run (including anywhere in their subhierarchy) have their original extents returned; all other cells are ignored.

### Description

Specifies to include shapes and text objects that occur in windows formed for each *cell\_name* during a Calibre run. This statement functions essentially like [Layout Window](#), but rather than providing coordinates, you provide cell names. For each cell C, the extents of all placements of C in the input layout database from the flat viewpoint are scanned. The coordinates of each extent are then used in an implicit Layout Window statement.

Calibre warns for each *cell\_name* argument that is not located in the input layout database.

If ORIGINAL is specified, then all objects in the input layout database are used to compute the specified cell extent(s), not just those objects actually required by the flow. If OCCUPIED is additionally specified, then all objects in the input layout database are used to compute the specified cell extent(s), if there is at least one object required by the flow in the cell or its subhierarchy; however, if a specified cell does not actually contain objects that are required by the flow (either in the cell or its sub-hierarchy), then the extent returned by this operation for that cell is empty (that is, as if ORIGINAL was not specified).

By default, the cell extents used by these statements are the extents of only those objects (text and geometry) actually required by the run. Calibre applications completely ignore objects in the input layout database that are not actually required by the flow. For example, if the input layout database contains objects on layer 53, but layer 53 is nowhere in a rule check or is not required connectivity in the current run, then this layer is ignored as the input layout database is

processed. Being ignored implies that objects on layer 53 are not used to compute the extents of individual cells.

See also [Layout Window Layer](#), [Layout Windel Cell](#), and [Layout Window Clip](#).

## Examples

### Example 1

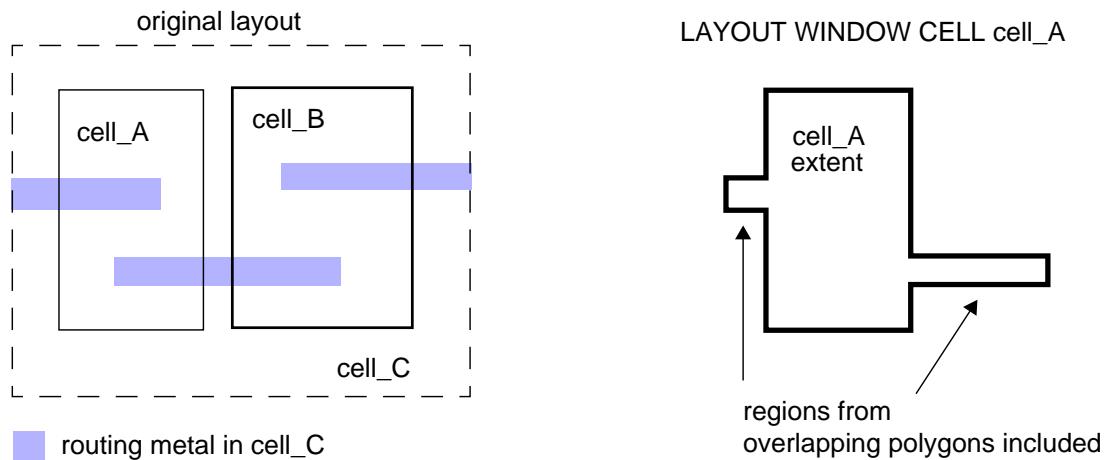
Using the ORIGINAL keyword:

```
Layout Window Cell "dff_*" memblock ORIGINAL
/* include extents of all dff_ cells and memblock for windows;
use all objects in input database for computing extents */
```

### Example 2

[Figure 4-150](#) shows a diagram of a Layout Window Cell region:

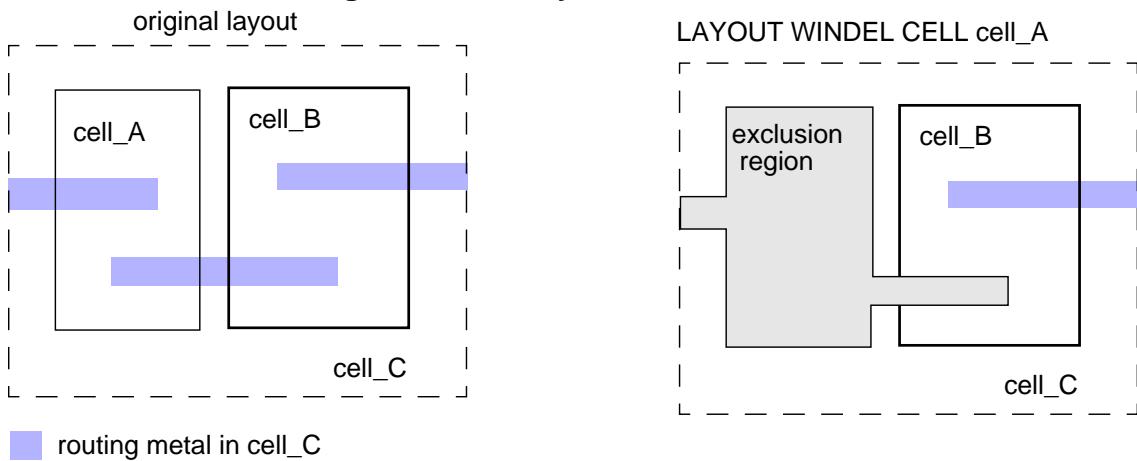
**Figure 4-150. Layout Window Cell**



### Example 3

[Figure 4-151](#) shows a diagram of a Layout Windel Cell exclusion region:

**Figure 4-151. Layout Windel Cell**



## Layout Window Clip

Specification statement

### LAYOUT WINDOW CLIP {NO | YES}

Used only in Calibre.

#### Parameters

- **NO**

Keyword that indicates no filtering for Layout Windel and Layout Window specification statements. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that indicates exclusive filtering for Layout Windel and Layout Window specification statements.

#### Description

Specifies whether layout object filtering is applied to [Layout Windel](#) and [Layout Window](#) statements in cases where layout objects share some, but not all of their area with the windows. The default is NO, which indicates no filtering. This statement can be specified once in your rule file.

When used with Layout Windel specification statements, Layout Window Clip YES discards portions of database objects lying *inside* data exclusion windows. When used with Layout Window specification statements, Layout Window Clip YES discards portions of database objects lying *outside* data inclusion windows. If neither Layout Windel nor Layout Window are specified, then this statement has no effect.

For hierarchical Calibre applications, the filtering is slightly more complicated: from the top-level down, placements whose extents cross a data inclusion window are flattened one level. The process is recursive until no placements cross the window; geometries and text objects are then filtered as discussed previously.

All input objects are filtered by layout inclusion windows, including polygons specified in [Polygon](#) or [Layout Polygon](#) specification statements and rule file text objects. In addition, the database extent (see [Extent](#)) is recomputed to account for any area filtering.

This statement also applies to [Layout Windel Cell](#), [Layout Windel Layer](#), [Layout Window Cell](#), and [Layout Window Layer](#).

#### Examples

For examples, refer to the [Layout Windel](#) and [Layout Window](#) specification statements.

# Layout Window Layer

Specification statement

**LAYOUT WINDOW LAYER** *layer* [*layer* ...]

Used only in Calibre.

## Parameters

- *layer*

Required original layer or layer set. Any number of layers may be specified.

## Description

Specifies to include layout objects that occur inside windows formed for each *layer* during a Calibre run.

This statement behaves essentially like [Layout Window](#), but rather than providing coordinates, you provide layer names. For each layer L, all shapes on L are scanned from the input layout database and transformed to top-level cell space. The coordinates of the vertices of each polygon on L are then used to form an implicit Layout Window statement.

This statement may be specified any number of times.

See also [Layout Window Cell](#), [Layout Windel Layer](#) and [Layout Window Clip](#).

## Examples

### Example 1

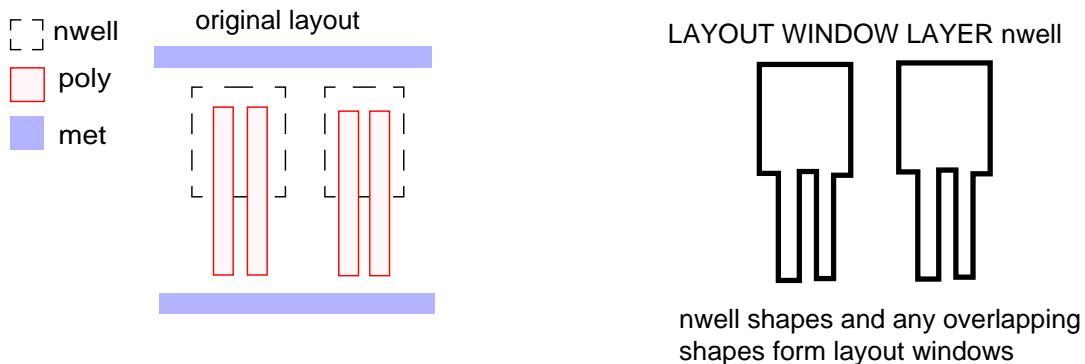
Basic syntax:

```
LAYOUT WINDOW LAYER nwell
/* use nwell shapes for windows, including overlapping polygons */
```

### Example 2

[Figure 4-152](#) shows a diagram of a Layout Window Layer region:

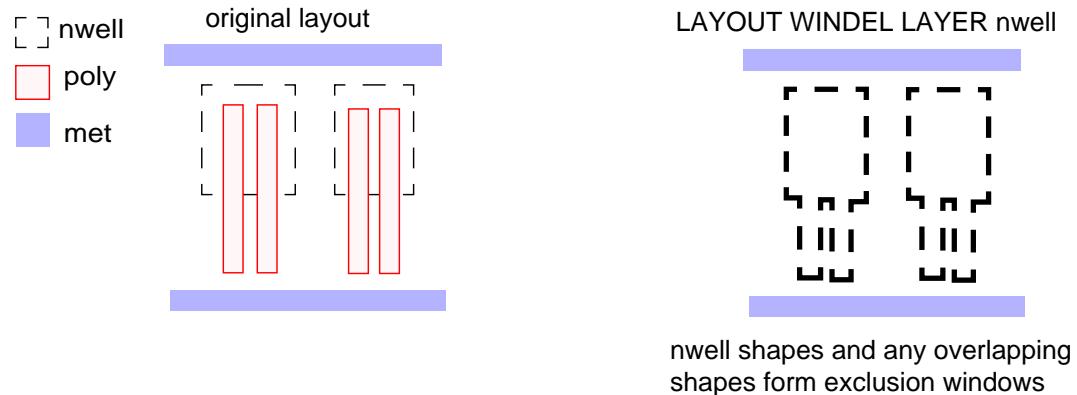
**Figure 4-152. Layout Window Layer**



### Example 3

Figure 4-153 shows a diagram of a Layout Windel Layer region:

**Figure 4-153. Layout Windel Layer**



# Length

Layer operation

## LENGTH *layer constraint*

### Parameters

- *layer*  
An original layer or layer set, or a derived polygon or edge layer.
- *constraint*  
A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.  
The constraint must contain non-negative real numbers, which are in user units.

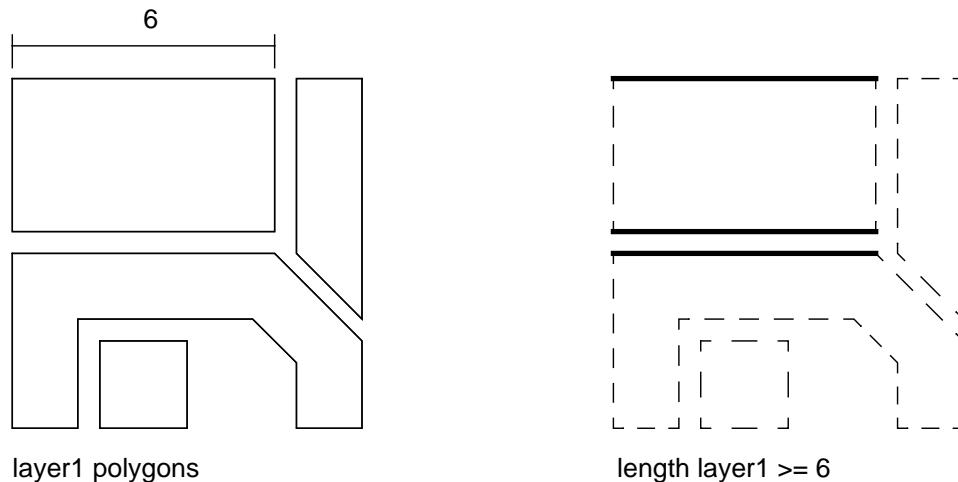
### Description

Selects all edges of *layer* having length that conforms to the *constraint*. See also [Not Length](#), [Path Length](#), and [Unit Length](#).

### Examples

The Length operation shown in Figure 4-154 selects all layer1 edges with length greater than or equal to 6 units.

**Figure 4-154. Length**



# Litho DenseOPC

Layer operation

**LITHO DENSEOPC** *layer* [*layer* ...] **FILE** {*filename* | *name* | '['  
*inline\_file*  
 ']'} **MAP** *name*

Used only in Calibre nmOPC applications.

## Parameters

- ***layer***

A required original or derived polygon layer upon which to perform dense OPC operations. You can specify *layer* any number of times in one statement.

- **FILE** {*filename* | *name* | '['  
*inline\_file*  
 ']'}

Required keyword, followed by one of the following:

A *filename* indicating the path to the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).“

The *name* parameter of a [Litho File](#) specification statement. The setup file is specified in the Litho File statement.

An *inline\_file* between brackets ([ ]). Characters within the brackets on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP** *name*

A required keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output.

## Description

Specifies to run Calibre nmOPC during Calibre nmDRC or nmDRC-H processing a *layer* or multiple layers you specify.

These applications output a new target layer containing OPC-corrected shapes based on the specified input layer(s) and the setlayer denseopc RET command.

This command can be specified any number of times in your rule file. By default, it produces a log file named litho.log in the working directory.

Refer to the [Calibre nmOPC User’s and Reference Manual](#) for process, procedures, and setlayer denseopc reference. The manual also describes the technology setup file.

## Examples

The following example performs OPC on the shapes in layer M1 using the MY\_DENSE setup file options. In the MY\_DENSE file occurs a setlayer denseopc command for the input layer, M1 along with many other settings and options (not shown). The output is written to the M1\_OPc layer, which is returned to LITHO DenseOPC and mapped to the M1\_bias2 layer. This layer is then made available to other Calibre operations using layer assignment (*derived\_layer* = ...).

```
M1_bias2 = LITHO DENSEOPC M1 FILE MY_DENSE MAP M1_bias2  
LITHO FILE MY_DENSE [  
    ...  
    setlayer M1_OPc = denseopc M1 sraf MAP m1 OPTIONS OPTS  
    ...  
]
```

## Litho File

Specification statement

```
LITHO FILE name '['  
    inline_file  
    ']'
```

Used only in Calibre RET and FRACTURE applications.

### Parameters

- *name*

A required name of an inline file to be searched for by a [Litho DenseOPC](#), [Litho OPC](#), [Litho OPCVerify](#), [Litho ORC](#), [Litho Printimage](#), [RET NMDPC](#), [RET PXOPC](#), [RET SBAR](#), [RET Sraf\\_Fill](#), [Fracture JEOL](#), [Fracture MEBES](#), [Fracture NUFLARE](#), [DBCLASSIFY](#), or [MDPverify](#) operation.

- '['  
 *inline\_file*  
 ']'

A required technology setup file placed between [ ] brackets. It may span multiple lines. It contains setup instructions for the lithography operation that calls the *name* in a FILE keyword group. Characters that occur between the brackets and on the identical lines as the brackets are ignored.

### Description

Specifies the FILE parameter of the Litho, Fracture, or MDPverify operation to be inline separate from the operation itself, yet still in the rule file.

The FILE parameter of an operation making a call to a Litho File may indicate a Litho File *name*. For example:

```
LITHO FILE setup.poly [  
iterations 5  
gridsize 0.00109375  
stepsize 0.00109375  
tilemicrons 80.000000  
sse OPC_MIN_EXTERNAL 0.075 0.075  
]  
  
LITHO OPC poly FILE setup.poly
```

When the above Litho operation is compiled, the rule file is searched for the Litho File specification statement with a *name* of setup.poly. (This search is case-insensitive.) If found, then setup.poly is used in the Litho operation. If not found, then setup.poly is assumed to be an external file.

No Litho File specification statements may share the same *name*.

# Litho OPC

Layer operation

**LITHO OPC** *layer* [*layer* ...] **FILE** {*filename* | *name* | ‘[  
*inline\_file*  
‘]’} [MAPNUMBER *value* | MAP *name*]

Used only in Calibre OPCpro applications.

## Parameters

- ***layer***

A required original layer, or derived polygon or edge layer, upon which to perform Calibre OPCpro operations. You can specify *layer* any number of times in one statement. The first *layer* is the target layer for correction. Any additional *layers* are supplementary (for example, non-printing islands).

- **FILE** {*filename* | *name* | ‘[  
*inline\_file*  
‘]’}

Required keyword, followed by one of the following:

A *filename* indicating the path to the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

The *name* parameter of a [Litho File](#) specification statement. The setup file is specified in the Litho File statement.

An *inline\_file* between brackets ([ ]). Characters within the brackets on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAPNUMBER** *value*

An optional keyword, followed by the tag layer to output. The parameter *value* is the layer specified in the setup file’s layer statement. Not used with the MAP keyword.

- **MAP** *name*

An optional keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output. Not used with MAPNUMBER.

## Description

Specifies to run Calibre OPCpro during Calibre nmDRC or nmDRC-H. The command takes in a target layer and outputs a new layer with OPC-corrected shapes.

This command can be specified any number of times in your rule file. By default, it produces a log file named litho.log in the working directory. It works in conjunction with an RET layer command in the setup file.

Refer to the [Calibre OPCpro User's Manual](#) for Calibre OPCpro process and procedures. The manual also describes the technology setup file.

### Examples

#### Example 1

This example shows how the *layer* in the SVRF call is matched by a layer command in the setup file. It uses the *inline\_file* syntax, but only shows the relevant setup instruction.

```
OPC_OUT = LITHO OPC poly FILE [
    ...
    layer 0 poly 0 0 opc dark
    ...
]
```

#### Example 2

This example shows the difference between using MAP and MAPNUMBER in LITHO OPC for PSM masks. Using MAP and the layer name has the advantage of using the layer name “PHASE0”, as opposed to the setup file map number.

Using MAP:

```
PHASE0_OPCT = LITHO OPC PHASE0 PHASE180 BLOCK TARGET FILE "setup.in"
    MAP PHASE0
```

Using MAPNUMBER:

```
PHASE0_OPCT = LITHO OPC PHASE0 PHASE180 BLOCK TARGET FILE "setup.in"
    MAPNUMBER 100
```

LITHO setup file for both cases:

```
layer 100 PHASE0 0 0 correction clear 1 1.0
```

# Litho OPCverify

Layer operation

**LITHO OPCVERIFY** *layer [layer ...] FILE {filename | name | '[' inline\_file ']' } MAP name*

Used only in Calibre OPCverify applications.

## Parameters

- ***layer***

One or more required layers sent to the OPCverify command file for processing. All layers are processed in the order listed.

- **FILE {filename | name | '[' inline\_file ']' }**

Required keyword, followed by one of the following:

A *filename* indicating the path to the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).“

The *name* parameter of a [Litho File](#) specification statement. The setup file is specified in the Litho File statement.

An *inline\_file* between brackets ([ ]). Characters within the brackets on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP *name***

A required keyword, followed by the name of a layer from the setup file. This layer is then mapped to the layer in the rule file for output.

## Description

Specifies to run Calibre OPCverify during Calibre nmDRC or nmDRC-H, simulating and checking the *layers* you specify. The setup file contains the instructions for simulating and processing, and passes them back using a command of the form “setlayer *name*”, where *name* matches the *name* in the Litho OPCverify command.

This command can be specified any number of times in your rule file. By default, it produces a log file named litho.log in the working directory.

Refer to the [Calibre OPCverify User’s and Reference Manual](#) for Calibre OPCverify process and procedures. The manual also describes the setup file commands.

## Examples

The following example shows how layers are mapped between the Litho OPCverify command and the setup file. (The contents of the setup file show only the parts relevant to this example. A working setup file would also specify several other settings.)

```
L1 = LITHO OPCVERIFY poly_opc sbars FILE setup.in MAP image1
L2 = LITHO OPCVERIFY poly_opc sbars FILE setup.in MAP image2
...
LITHO FILE setup.in [
    ...
    layer mask visible dark
    layer sraf visible dark
    ...
    setlayer image1 = image optical m1 aerial_contour 0.3
    setlayer image2 = image optical m1 aerial_contour 0.4
    ...
]
```

The Litho OPCverify commands are identical except for the output layer specified by the MAP keyword. The commands pass the poly\_opc and sbars layers to the setup file, where they are mapped to the mask and sraf layers respectively.

In the setup file, the commands “image optical...” produce the simulations and generate a layer. The first one is written to a layer named image1 and the second to image2. The MAP keyword in the Litho OPCverify statement identifies the intended matching SVRF statement.

In this example, the Litho OPCverify statements then assign the output to a layer available to the rest of Calibre. This makes it available for more processing. Alternatively, the Litho OPCverify statement might be enclosed in a rule check and have their output written directly to a GDS or results database.

# Litho ORC

Layer operation

**LITHO ORC FILE** {*filename* | *name* | '['  
*inline\_file*  
 ']'} **layer\_in** [*layer\_aux* ...] [**MAPNUMBER** *value* | **MAP** *name*]

Used only in Calibre ORC applications.

## Summary

Layer operation used to invoke Calibre ORC for tagging edges or verifying OPC results.

## Parameters

- **FILE** {*filename* | *name* | '['  
*inline\_file*  
 ']'}

Required keyword, followed by one of the following:

A *filename* indicating the path to the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).“

The *name* parameter of a [Litho File](#) specification statement. The setup file is specified in the Litho File statement.

An *inline\_file* between brackets ([ ]). Characters within the brackets on the same line as them are ignored. Inline files contain setup instructions for OPC and ORC and span multiple lines. The instructions may not use environment variables.

- **layer\_in**

A required original or derived polygon layer that serves as input. In the setup file, it must have type opc.

- **layer\_aux**

An optional layer or layers that provides additional simulation or fragmentation data. In the setup file, these layers can be of type visible, asraf, external, or island.

- **MAPNUMBER** *value*

An optional keyword followed by the tag layer to output. The parameter *value* is the layer specified in the technology setup file with the tags2boxes command for ORC applications. Not used with the MAP keyword.

- **MAP** *name*

An optional keyword, followed by the name of a layer from the technology setup file. This layer is then mapped to the layer in the rule file for output. Not used with MAPNUMBER.

### Description

Runs optical rule checking (ORC) during Calibre nmDRC or nmDRC-H on a single layer, *layer\_in*, and outputs a new nmDRC layer containing edges that have either been tagged or are in error.

This command can be specified any number of times in your rule file. By default, it produces a log file named litho.log in the working directory.

Refer to the [\*Calibre OPCpro User's Manual\*](#) for Calibre ORC process and procedures. The manual also describes the technology setup file.

### Examples

In the following example, POLY is the input layer to Calibre ORC and orcpoly.in is the technology setup file. Calibre ORC derives the layer MY\_ORC, which contains poly edges that were in error. This derived layer is then output using the ORC rule check and sent to a GDSII database file using DRC Check Map.

```
MY_ORC = LITHO ORC FILE orcpoly.in POLY
ORC.1 { copy MY_ORC }
DRC CHECK MAP ORC.1 orc.gds GDSII
```

# Litho Printimage

Layer operation

**LITHO PRINTIMAGE FILE** {*filename* | *name* | '['  
*inline\_file*  
 ']' } *layer* [*layer* ...] [MAPNUMBER *value*]

Used only in Calibre PRINTImage applications.

## Parameters

- **FILE** {*filename* | *name* | '['  
*inline\_file*  
 ']' }

Required keyword, followed by one of the following:

A *filename* indicating the path to the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).“

The *name* parameter of a [Litho File](#) specification statement. The setup file is specified in the Litho File statement.

An *inline\_file* between brackets ([ ]). Characters within the brackets on the same line as them are ignored. Inline files contain setup instructions and span multiple lines. The instructions may not use environment variables.

- ***layer***

A required original or derived polygon layer upon which to perform Calibre PRINTImage operations. The first layer is the target layer and the optional layers are supplementary.

- **MAPNUMBER *value***

An optional keyword followed by the contour level to output. If this keyword is not used then all contours are returned in a single layer.

## Description

Invokes Calibre PRINTImage during a Calibre nmDRC or nmDRC-H run. Calibre PRINTImage simulates a scanning electron microscope (SEM) image of the polygons on the first *layer*. The simulation data is generated as a set of contours.

Generally Litho Printimage commands provide at least two *layers*: the target layer and an OPC-corrected layer. In the setup file, the target layer must be of type opc and the corrected layer must be of type external. Syntactically, the first layer must always be of type opc and any other layers must be of type visible, asraf, external, or island. The supplemental layers cannot be hidden, correction, or opc types.

The Litho Printimage command can be specified any number of times in your rule file. By default, it produces a log file named litho.log in the working directory.

Refer to the [\*Calibre OPCpro User's Manual\*](#) for Calibre OPCpro and Calibre PRINTImage process and procedures.

### Examples

This example generates an SEM image for a simple binary mask. The Litho Printimage command is part of a DRC rule check.

```
MY_SEM { LITHO PRINTIMAGE FILE printimage.in POLY }
DRC CHECK MAP MY_SEM 3

LITHO FILE printimage.in [
    ...
    layer 2 poly 0 0 opc dark
    ...
]
```

Because the example provides only the target layer, the simulated image does not reflect OPC correction.

# Litho PSMgate

Layer operation

## **LITHO PSMGATE *phase\_layer feature\_layer***

Used only in Calibre RET applications.

### Parameters

- ***phase\_layer***  
A required layer whose shapes indicate the regions for which to calculate phase-shifted output.
- ***feature\_layer***  
A required layer containing actual target features.

### Description

Runs Calibre PSMgate to produce phase-shifted masks.

This command can be specified any number of times in your rule file. By default, it produces a log file named litho.log in the working directory.

### Examples

This example shows calculating a two-phase mask.

```
gate = POLY AND DIFF
sd = DIFF NOT gate

//Phase assignment
PHASE1 = LITHO PSMGATE diff gate

//Form Phase2 as the complement of Phase1
PHASE2 = sd NOT PHASE1
```

## LVS Abort On ERC Error

Specification statement

### LVS ABORT ON ERC ERROR {YES | NO}

Used only in Calibre nmLVS/nmLVS-H and PERC.

#### Parameters

- **YES**

A required keyword that allows ABORT LVS keywords specified in [ERC Select Check](#) statements to terminate nmLVS. This is the default behavior if this statement is not specified in your rule file.

- **NO**

A required keyword that does not allow ABORT LVS keywords specified in ERC Select Check statements to terminate nmLVS.

#### Description

Controls LVS termination when ABORT LVS keywords are specified in ERC Select Check statements. This statement is useful for quickly disabling all ABORT LVS options instead of removing them from each ERC Select Check statement in your rule file.

If YES is specified (the default), then LVS terminates if ABORT LVS is used in a ERC Select Check statement and a specified rule check returns non-empty results. Warnings are written in the run transcript for each selected ERC check that reports an error. Connectivity extraction and device recognition occur as usual, but LVS terminates before the circuit comparison stage.

If NO is specified, LVS ignores any ABORT LVS keywords used in ERC Select Check statements. No special messages are written to the transcript if NO is specified.

This statement may be specified once in your rule file.

#### Examples

```
// Observe ABORT LVS keywords in ERC Select Check statements
LVS ABORT ON ERC ERROR YES
```

## LVS Abort On Softchk

Specification statement

### LVS ABORT ON SOFTCHK {NO | YES}

#### Parameters

- **NO**

Keyword that instructs the tool *not* to abort processing when a violation is detected. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to abort processing when a violation is detected.

#### Description

Specifies whether to abort processing when the tool finds a soft connection conflict. LVS aborts after reporting any [LVS Softchk](#) errors (when LVS Softchk statements are specified) and after executing any ERC operations and generating the ERC output. LVS aborts if [Sconnect](#) conflicts are present regardless of whether you specified LVS Softchk statements.

LVS does not abort the process until the circuit extraction phase is complete and before the circuit comparison stage begins.

YES is most frequently specified when it is desirable that shorts through well regions cause LVS to abort.

You can specify soft connection management options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Aborting on Sconnect Errors](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [LVS Report Option S](#).

#### Examples

```
// Do not abort LVS on detection of soft connection.  
LVS ABORT ON SOFTCHK NO
```

## LVS Abort On Supply Error

Specification statement

### LVS ABORT ON SUPPLY ERROR {YES | NO}

#### Parameters

- **YES**

Keyword that instructs nmLVS to abort processing after connectivity extraction detects a problem with any of the power or ground nets. This is the default behavior for flat nmLVS.

- **NO**

Keyword that instructs nmLVS not to abort processing when connectivity extraction detects a problem with any of the power or ground nets. This is the default behavior for nmLVS-H.

#### Description

Specifies whether to abort Calibre nmLVS processing when connectivity extraction detects a problem with any of the power or ground nets appearing in the [LVS Power Name](#) and [LVS Ground Name](#) specification statements. Problems that cause processing to abort include these:

- nmLVS did not assign a power or ground name to a net because it conflicted with another name that was found on the net (a short).
- nmLVS found a single power or ground name on two disjoint nets (an open).
- nmLVS found an unattached power or ground name.
- nmLVS found an ambiguously named top-level port.
- nmLVS found an error in a [Stamp](#) operation involving a power or ground net.

These problems generate warnings during the run.

When YES is specified, such supply problems cause the run to stop after completing the circuit extraction step (including all processes performed in this step like netlist extraction, ERC, short isolation, and so forth). If such occurs, the following error is written in the transcript:

```
ERROR: Supply error detected. ABORT ON SUPPLY ERROR is specified -  
aborting.
```

In hierarchical runs, only supply net names in the top-level cell are considered. Connectivity problems must affect the top-level cell in order for the run to abort. The YES option only applies when connectivity extraction and circuit comparison are specified in a single run. That is, the -lvs -hier -spice command-line options are specified; or -lvs -hier is specified, the [Layout System](#) is geometric, and the [Mask SVDB Directory](#) statement is specified. If the run aborts, the comparison step is not performed.

The NO option is most frequently specified when debugging issues involving power and ground nets.

You can specify power supply management options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Setting Power Supply Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## Examples

```
// Abort LVS before the comparison phase upon detection of a supply error.  
LVS ABORT ON SUPPLY ERROR YES
```

## LVS All Capacitor Pins Swappable

Specification statement

**LVS ALL CAPACITOR PINS SWAPPABLE {NO | YES}**

### Parameters

- **NO**

Keyword that specifies that all capacitor pins should *not* be considered swappable. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that specifies that all capacitor pins should be considered swappable

### Description

Specifies whether all capacitor pins should be considered swappable. This statement can be specified once in your rule file. In Pyxis Layout, this statement sets the value of application variable lvs\_all\_capacitor\_pins\_swappable.

### Examples

```
// Do not allow cap pins to be swappable.  
LVS ALL CAPACITOR PINS SWAPPABLE NO
```

# LVS Annotate Devices

Specification statement

## LVS ANNOTATE DEVICES *device\_layer*

Used only in Calibre nmLVS-H and PERC.

### Parameters

- *device\_layer*

A required layer that must be derived from a [Device Layer](#) ANNOTATE operation.

### Description

Specifies whether to annotate devices in the extracted layout netlist with DFM properties taken from the *device\_layer*. The *device\_layer* must touch with the appropriate device seed layer to have the properties annotated to the device. This statement is used during connectivity extraction. It may be specified any number of times with unique *device\_layer* parameters.

DFM Property names starting with the string ICV\_ are reserved for internal use of the tool and are not copied to devices by this statement.

Use of this statement follows this flow:

1. Generate measurements of interest for a particular device and apply them with the [DFM Property](#) operation. These measurements might include, for example, external measurements from a device gate polygon to nearby gate polygons. See “[Using DFM Property with Device Layer ANNOTATE Programs](#)” on page 1848 for an example.
2. Generate a layer using the Device Layer ANNOTATE operation. This creates a layer containing recognized device seed shapes that are annotated with all NUMBER, STRING, and DFM VECTOR properties from the ANNOTATE layer, as well as device-specific information that can be used by the Query Server’s CCI interface to create netlists that have additional properties that were not present in the original LVS netlist.
3. Pass the output from Device Layer as the *device\_layer* parameter to LVS Annotate Devices. This is stored in the PHDB database of the [Mask SVDB Directory](#) (using the CCI option).
4. Run connectivity extraction to have the properties annotated to the appropriate devices.
5. Run the Calibre Connectivity Interface annotated device commands to produce layout annotated SPICE netlists or annotated GDS files (AGF) that contain the properties. See “[Annotated Device Commands](#)” in the *Calibre Query Server Manual*.

## Examples

Consider the following Device statement:

```
// MP Device:  
DEVICE MP PGATE PGATE(g) SD(s) SD(d) NW(b) [  
    property w,l  
    w=perimeter_co(PGATE,SD) * 0.5  
    l=perimeter_outside(PGATE,SD) * 0.5  
]
```

To annotate the area of overlapping polygons from layer L1 onto the device, the following sequence of layer operations is required:

```
// Create a layer that has L1 area annotated onto pgate polygons:  
pgate_with_area = DFM PROPERTY pgate L1 OVERLAP ABUT ALSO MULTI  
    [ A_L1 = AREA( L1 ) ]  
  
// Generate the annotated device layer from MP devices that interact with  
// pgate_with_area  
dev_layer_mp = DEVICE LAYER MP ANNOTATE pgate_with_area  
  
// Pass device property layer on to LVS ANNOTATE DEVICES.  
LVS ANNOTATE DEVICES dev_layer_mp  
  
MASK SVDB DIRECTORY svdb QUERY CCI
```

MP devices that interact with dev\_layer\_mp will have the property A\_L1 associated with them.

Annotated property layers can be accessed using Calibre Connectivity Interface annotated device commands. The layers can be used for custom layout SPICE netlist generation and annotated GDS file (AGF) generation. Here is an example CCI script:

```
GDS ANNOTATED DEVICES YES  
GDS WRITE annotated_MN.agds  
LAYOUT NETLIST ANNOTATED DEVICES YES  
LAYOUT NETLIST WRITE annotated_MN.sp  
TERMINATE
```

## LVS Auto Expand Hcells

Specification statement

**LVS AUTO EXPAND HCELL[S] {NONE | PRESET | ALL | *expansion\_threshold*}**  
**[REPORT {PRESET | ALL | NONE | *report\_threshold*}]**

Used only in hierarchical connectivity extraction.

### Parameters

- **NONE**

Keyword that specifies no hcells are expanded that are subject to the optimization shown in the HIGH-COST HCELLS transcript report (if any). This is the default behavior for hcell expansion.

- **PRESET**

Keyword that specifies the tool automatically selects which hcells to expand. Expanded hcells are subject to the optimization shown in the HIGH-COST HCELLS transcript report (if any).

- **ALL**

Keyword that specifies all hcells are expanded if they are subject to the optimization shown in the HIGH-COST HCELLS transcript report (if any).

- ***expansion\_threshold***

A positive floating-point number less than 100 that specifies a minimum threshold for hcell expansion. Hcells that exceed this threshold for the optimization shown in the HIGH-COST HCELLS transcript report (if any) are expanded.

- **REPORT { PRESET | ALL | NONE | *report\_threshold* }**

Optional keyword set that controls the reporting of HIGH-COST HCELLS in the run transcript. The REPORT options must be specified after the other parameters in the command. The REPORT keyword takes these additional arguments:

PRESET — Specifies that the tool automatically selects which hcells' scores to report. This is the default for reporting.

ALL — Specifies to report all hcells that have a performance cost score.

NONE — Specifies not to report hcell performance cost scores.

*report\_threshold* — A positive floating-point number less than 100 that specifies a minimum threshold for reporting of hcell performance cost scores.

### Description

Specifies whether to expand hcells automatically during Calibre nmLVS-H connectivity extraction (calibre -spice) in order to optimize performance. Additionally, this statement controls the automatic reporting of hcell performance cost scores in the run transcript based upon certain hierarchy modifications performed automatically by the tool. See the “[Hcells in](#)

[“LVS Circuit Extraction”](#) section of the *Calibre Verification User’s Manual* for a discussion of which hierarchy management optimizations are subject to this specification statement. This statement has no effect during the LVS comparison phase and may be specified once in the rule file.

Circuit extraction in Calibre nmLVS-H can be adversely affected by declaring some cells as hcells. Hcells are not subject to most hierarchy optimizations and transformations performed by Calibre nmLVS-H. Such transformations are intended to improve performance and scaling of geometry processing. This specification statement enables you to improve connectivity extraction performance by managing hcell expansion.

Calibre nmLVS-H employs an hcell performance cost analyzer. The results of the hcell cost analysis are reported in the nmLVS-H connectivity extraction transcript. During the hierarchical database construction phase, hcell performance cost is reported in the section entitled **HIGH-COST HCELLS**. The estimated performance impact of an hcell is measured and rated between 0 to 100, with 100 representing the worst case. The following example shows how such a report might appear:

```
HIGH-COST HCELLS
blk1 (EXPANDING DENSE OVERLAPS): 73.8
route2 (EXPANDING DENSE OVERLAPS): 70.8
blk2 (EXPANDING DENSE OVERLAPS): 51.2
seg (EXPANDING DENSE OVERLAPS): 25.7
```

The report shows the names of the hcells, the hierarchy-management optimization for which the cost is being estimated, and the cost score. If an hcell is not subject to any optimizations considered by this statement, it is not reported. If an hcell is not subject to any of the optimizations that may appear in the report, it is not expanded.

The first set of keywords for this statement controls whether automatic hcell expansion occurs. One of these keywords must be specified when this statement appears in the rule file. The **NONE** keyword is used by default and expansion does not occur. Specifying **PRESET** causes the tool to select which hcells to expand automatically. Specifying an *expansion\_threshold* causes the tool to expand hcells whose cost score is greater than the specified threshold. Specifying **ALL** causes all hcells to be expanded. The **NONE** and **ALL** keywords can be thought of as representing the threshold values 100 and 0, respectively.

If an hcell is automatically expanded, then the following message appears in the transcript:

```
HCELL blk1 REJECTED, COST OF EXPANDING DENSE OVERLAPS: 73.8
```

The **REPORT** set of keywords specifies which hcells get reported in the **HIGH-COST HCELLS** section of the transcript. This set of keywords is optional. By default, the tool uses the **PRESET** option and automatically determines which hcells to report. Specifying **REPORT report\_threshold** causes the tool to report hcells whose cost value is greater than the specified threshold. Specifying **REPORT ALL** causes all cost scores to be reported. Specifying **REPORT NONE** causes no cost scores to be reported.

The cost scores for hcells can vary across releases due to optimizations in the hierarchical database constructor. This could cause changes in the expansion and reporting of specific hcells for a given threshold value.

Hcells specified in [Layout Preserve Cell List](#) statements are subject to expansion by the LVS Auto Expand Hcells statement.

LVS Box cells are never expanded by the LVS Auto Expand Hcell statement. Similarly, Calibre xcells (in a list specified with the -xcell option) are not expanded when they also serve as hcells. This applies to layout hcells and xcells only. Hcells that are either box cells or xcells still appear in the HIGH-COST HCELL report, however. See “[Controlling Hierarchy with Xcells](#)” in the *Calibre xRC User’s Manual* for details about xcell files.

See “[LVS Connectivity Extraction](#)” and “[Hierarchical Circuit Extraction](#)” in the *Calibre Verification User’s Manual* for detailed information about hierarchical connectivity extraction.

See also [Hcell](#) and [LVS Exclude Hcell](#).

## Examples

### Example 1

This example specifies that hcells are expanded automatically as the tool determines is best for performance. Reporting is according to the default setting. This is the recommended specification for most situations.

```
LVS AUTO EXPAND HCELL PRESET
```

### Example 2

This example specifies that no hcells are expanded automatically, and hcells with a cost score greater than 25 are reported.

```
LVS AUTO EXPAND HCELL NONE REPORT 25
```

### Example 3

This example specifies that all hcells having a cost score greater than 25 are both expanded and reported.

```
LVS AUTO EXPAND HCELL 25 REPORT 25
```

### Example 4

This example specifies that all hcells having a cost score greater than 25 are expanded. The reporting of cost scores is turned off.

```
LVS AUTO EXPAND HCELL 25 REPORT NONE
```

## LVS Box

Specification statement

**LVS BOX** [SOURCE] [LAYOUT] *cell* [*cell* ...]

Used only in Calibre nmLVS/nmLVS-H, PERC, and xRC.

### Parameters

- SOURCE  
An optional keyword that specifies the statement applies to the source design. This option is used by default.
- LAYOUT  
An optional keyword that specifies the statement applies to the layout design. This option is used by default.
- *cell*  
A required name of a cell. You can specify *cell* any number of times in one statement. A *cell* parameter may be specified as a string [Variable](#).

### Description

Specifies *black box* cells in hierarchical circuit extraction (calibre -spice) as well as flat and hierarchical circuit comparison.

This statement can be specified any number of times. Layout and source specifiers are accumulated per cell. Empty subcircuit names can be specified.

### Hierarchical Circuit Comparison

The specified cells are boxed out in the circuit comparison stage of hierarchical nmLVS (calibre -lvs -hier). Calibre nmLVS-H ignores the contents of the specified cells when it reads the layout or source netlists and treats these cells as primitive devices with pins.

The LVS component type of box cells is the cell name. The cell has no subtype, but subtypes may be specified with the \$[mname] or \$.MODEL = mname constructs in subcircuit calls in SPICE netlists, just like for empty subcircuits. LVS pin names are the cell pin names.

Like all primitive devices, box cells are matched by name by default. The [Hcell](#) specification statement or an hcell file must be used to specify different cell matching. For example, if box cell A matches cell A\_src, you must specify hcell correspondence and must use:

LVS BOX A\_lay A\_src  
or

```
LVS BOX LAYOUT A_lay
LVS BOX SOURCE A_src
```

The .GLOBAL and .PARAM specifications in SPICE netlists are processed even when they appear within box subcircuits.

Specifying top cells as box cells causes the designs to appear as empty.

**High-short Resolution in LVS Box Cells** — See “[High-short Resolution](#)” in the *Calibre Verification User’s Manual*.

## Hierarchical Circuit Extraction

The LVS Box statement ensures that all information necessary to process box cells in the comparison stage is preserved in the extraction stage.

**Netlisting** — All placements of layout box cells are written as subcircuit calls to the extracted layout netlist. Normally, the hierarchical SPICE netlister discards placements of empty cells (that is, cells with no devices). This filtering is disabled for layout box cells.

The netlisting of box cell contents is controlled by the [LVS Netlist Box Contents](#) statement. By default, box cell contents are netlisted. You can suppress this by specifying LVS Netlist Box Contents NO. The [LVS Preserve Box Cells](#) YES statement has the same effect. Additionally, LVS Preserve Box Contents YES suppresses seed promotion out of box cells and the netlisting of any devices that originate from within box cells.

All named (texted) pins of layout box cells are written to the extracted layout netlist. Recall that a pin name is simply the name of its respective net in the cell. A pin is named if the respective net is named. In addition, all pins of layout box cells that are connected to named (texted) port objects inside the box cell are written to the extracted layout netlist, except when the respective net inside the box cell is unnamed and [LVS Netlist Unnamed Box Pins](#) NO is specified.

Normally, the hierarchical SPICE netlister discards cell pins that are not connected to any devices or box placements within the cell. This filtering is disabled for pins in layout box cells that meet the above criteria.

You can instruct the hierarchical SPICE netlister to omit unnamed nets within LVS Box cells from the netlist. Such nets can be promoted to ports if they are connected to devices within the box cell and LVS Netlist Unnamed Box Pins YES is specified, which it is by default. You omit the unnamed nets by specifying LVS Netlist Unnamed Box Pins NO.

Named port objects can be created with [Port Layer Text](#) specification statements. To qualify, the port objects must be at the top level of the box cell; the port names must be non-null. Port objects in the sub-hierarchy of the box cell do not qualify. Note that unnamed port objects, such as those created by [Port Layer Polygon](#), do not qualify.

**Explicit Pins** — Port objects from Port Layer Text and Port Layer Polygon specification statements induce explicit pins in layout box cells. Specifically, any net in a layout box cell that has a port object connected to it in the cell becomes a pin of the cell. Thus, port objects can be used in box cells to ensure that a pin is recognized, even when the pin is floating in all placements of the cell. Note that if Port Layer Polygon is used for this purpose, the pin may not appear in the extracted layout netlist unless it meets other criteria. For example, the pin must be named, or must be connected to devices or box placements inside the box, or must be connected to named port objects, and so forth.

Note that normally there is no need to use port objects to specify pins in box cells. As always, any net in the cell that is connected to nets outside of the cell, in at least one placement of the cell, becomes a pin of the cell. Port objects can be used in box cells to specify pins explicitly that are globally floating and normally would not be recognized. Note also that port objects can

## LVS Box

---

induce pins but do not determine pin names or net names in box cells. Use regular [Text Layer](#) statements for texting.

**Cell Preservation** — Layout LVS Box cells are preserved during hierarchical netlist extraction. Layout LVS Box cells are not expanded by Calibre cell-expansion heuristics such as dense overlap removal, even if they otherwise fit those heuristics. This ensures that all layout LVS Box cells are preserved as subcircuits in the extracted layout netlist and are available for processing downstream. In this respect, layout LVS Box cells behave similarly to hcells. This special treatment applies to layout LVS Box cells and does not apply to LVS Box cells that are exclusively in the source set. Explicit user directives such as the specification statements [Expand Cell](#) and [Flatten Cell](#) have precedence and will expand LVS Box cells just like regular cells. Note that a LVS Box specification may slow down circuit extraction in cases where it would be beneficial to expand those LVS Box cells.

The LVS Netlist Box Contents NO and LVS Preserve Box Cells YES statements cause LVS Box cells to be written as empty subcircuits in the extracted netlist.

**Cell Pushdown** — Placements of layout LVS Box cells are not pushed down into other cells. Cells that are classified as very small can be pushed down as an optimization by the hierarchical database constructor. LVS Box cells do not participate in this.

## Flat LVS

In flat LVS, the LVS Box specification statement operates on the layout or source only if the layout or source, respectively, are SPICE netlists. The effect is then the same as described for hierarchical circuit comparison. Namely, LVS ignores the contents of the specified cells when it reads the layout or source netlists and treats these cells as primitive devices with pins. The layout or source are not affected if they are not in SPICE netlist form.

See also [LVS Preserve Box Ports](#), [LVS Report Option BX](#), and [LVS Spice Cull Primitive Subcircuits](#).

## Examples

### Example 1

In the following equivalent statements, the specified cells are processed as black boxes in the source and layout:

```
LVS BOX cella cellb cellc celld  
or
```

```
LVS BOX cella  
LVS BOX cellb  
LVS BOX cellc  
LVS BOX celld
```

or

```
LVS BOX layout cella cellb cellc celld  
LVS BOX source cella cellb cellc celld
```

**Example 2**

In the following statements, cella and cellb are boxed out in the layout only, and cellc and celld are boxed out in the source only:

```
LVS BOX layout cella cellb  
LVS BOX source cellc celld
```

## LVS Builtin Device Pin Swap

Specification statement

### LVS BUILTIN DEVICE PIN SWAP {YES|NO}

#### Parameters

- **YES**

Keyword that instructs the tool to allow default swapping of built-in device pins. This is the default behavior if you do not specify this statement.

- **NO**

Keyword that instructs the tool not to allow default swapping of built-in device pins.

#### Description

Specifies whether LVS should apply default pin swappability rules in built-in devices. Built-in devices are discussed in the [Device](#) section. This statement *does not* affect the operation of the [LVS All Capacitor Pins Swappable](#) specification statement.

Specifically, when YES is indicated, or when the statement is omitted, LVS applies the following default pin swappability rules:

- Source (S) and drain (D) pins of regular MOS transistors are considered logically equivalent. Regular MOS transistors are component types MN, MP, ME, MD, and M, and any equivalent types indicated with [LVS Device Type](#) specification statements.
- POS and NEG pins of resistor devices are considered logically equivalent. Resistor devices are component type R and any equivalent types indicated with LVS Device Type specification statements.

When NO is indicated, the default pin swappability rules do not apply. (Pin swappability is still inherited from respective rule file DEVice operations, if present, or may be specified by other means as described in the Device section). Source and drain pins in built-in MOS devices, and POS and NEG pins for R devices are always swappable if they are on the same layer, regardless of whether NO is specified.

**Gate recognition** — When NO is specified, regular MOS devices (component types MN, MP, ME, MD, and M) may have non-swappable source-drain pins. For the purpose of logic gate recognition, such devices are treated as LDD-type devices (component types LDDN, LDDP, LDDE, LDDD, and LDD respectively) and they form logic gates according to the rules that govern LDD-type devices. Logic gate naming convention is generally the same as for LDD gates, except that for non-voltage gates the generic naming convention is used, as follows:

S(TTT)n  
SP(TTT)\_n1\_n2\_...\_nm  
SP(TTT)(expression)

where the string “TTT” stands for the actual component type of the devices forming the gate.

This statement may be specified at most once.

## Examples

### Example 1

Consider the following DEVice operations:

```
DEVICE MP    PGATE1 POLY(G) PSRC(S) PDRN(D) NWELL(B)
DEVICE M(A)  PGATE2 POLY(G) PSRC(S) PDRN(D) NWELL(B)
DEVICE R     RESI L1(POS) L2(NEG)
```

Note that source and drain pins in the MP and M(A) devices are formed from different layers (PSRC for source and PDRN for drain). Also, POS and NEG pins in the R devices are formed from different layers (L1 and L2 respectively).

Normally, because of default pin-swappability rules, source and drain pins in the MP and M(A) devices would be swappable and POS and NEG pins in the R devices would also be swappable. If LVS Builtin Device Pin Swap NO is specified, then those pins are not swappable.

### Example 2

Consider the following DEVice operations:

```
DEVICE MP    PGATE1 POLY(G) PSD(S) PSD(D) NWELL(B)
DEVICE M(A)  PGATE2 POLY(G) PSD(S) PSD(D) NWELL(B)
DEVICE R     RESI L(POS) L(NEG)
```

Here, source and drain pins in the MP and M(A) devices are swappable and POS and NEG pins in the R devices are also swappable, regardless of the LVS Builtin Device Pin Swap specification statement. That is because these pins are formed from the same layers, respectively (PSD for both source and drain and L for both POS and NEG).

### Example 3

Consider the following:

```
DEVICE M(A) PGATE POLY(G) PSRC(S) PDRN(D) NWELL(B)
LVS BUILTIN DEVICE PIN SWAP NO
```

A logic gate consisting of three M(A) devices in series would be represented in the LVS report, as follows:

S(M)3

## LVS Builtin MOS NRD\_NRS

Specification statement

### LVS BUILTIN MOS NRD\_NRS {NO | YES}

#### Parameters

- **NO**

Keyword that specifies NRD and NRS properties are not swappable for parallel MOS device reduction. This is the default behavior.

- **YES**

Keyword that specifies NRD and NRS properties are swappable for parallel MOS device reduction.

#### Description

Specifies whether to treat NRS and NRD properties in MOS devices as swappable during LVS comparison. The NRS and NRD properties must be calculated in a *user-defined effective property calculation* in a [LVS Reduce Parallel MOS YES](#) or [LVS Reduce ... PARALLEL YES](#) for MOS devices specification statement. No built-in effective property calculations are made for these properties.

When NO is specified either explicitly or by default, NRS and NRD properties are not treated as built-in for comparison. This means these properties are not swappable when their associated devices are reduced.

When YES is specified, the NRS and NRD properties are swappable when their associated devices are reduced.

#### Examples

The following code assumes the Device statements for MOS devices calculate W, L, AS, AD, PS, PD, NRD, and NRS.

```
LVS REDUCE PARALLEL MOS YES
// Reduce MOS devices in parallel.
[ effective W, L, AS, AD, PS, PD, NRD, NRS
  // Calculate these effective values.
  P = sum( W * L ) // Sum of Wi * Li
  Q = sum( W / L ) // Sum of Wi / Li
  W = sqrt( P * Q ) // Effective W
  L = sqrt( P / Q ) // Effective L
  AS = sum( AS )    // Effective AS: sum of ASi
  AD = sum( AD )    // Effective AD: sum of ADi
  PS = sum( PS )    // Effective PS: sum of PSi
  PD = sum( PD )    // Effective PD: sum of PDi
  NRD = // Effective NRD calculation
  NRS = // Effective NRS calculation
]

LVS REDUCE SPLIT GATES YES
// Reduce split gate devices in parallel.
[ effective W, L, AS, AD, PS, PD, NRD, NRS
```

```
// Calculate these effective values.  
P = sum( W * L ) // Sum of Wi * Li  
Q = sum( W / L ) // Sum of Wi / Li  
L = sqrt( P / Q ) // Effective L  
W = sqrt( P * Q ) // Effective W  
AS = sum( AS ) // Effective AS: sum of ASi  
AD = sum( AD ) // Effective AD: sum of ADi  
PS = sum( PS ) // Effective PS: sum of PSi  
PD = sum( PD ) // Effective PD: sum of PDi  
NRD = // Effective NRD calculation  
NRS = // Effective NRS calculation  
]  
  
LVS BUILTIN MOS_NRD_NRS YES // allow swapping of NRD/NRS properties
```

## LVS Cell List

Specification statement

**LVS CELL LIST** *list\_name* *cell\_name* [*cell\_name* ...]

Used only in Calibre nmLVS-H and PERC.

### Parameters

- *list\_name*

An arbitrary name of a cell list.

- *cell\_name*

The name of an hcell (see the [Hcell](#) statement) that is in the source design or a non-hcell that is not in the source design. More than one cell name may be specified for a given list, and a cell name may only appear in a single list. A cell parameter may contain one or more asterisk (\*) characters; the \* character is a wildcard that matches zero or more characters. A string that contains a \* character must be enclosed in quotation marks because the \* is a reserved symbol.

### Description

Indicates a list of cells for writing cell-specific LVS rules.

Cell-specific LVS rules, also called *cell overrides*, are written by adding a cell *list\_name* argument to LVS specification statements that allow cell overrides. The [LVS Recognize Gates](#), [LVS Reduce](#), and [Trace Property](#) statements allow this. Generally, a *list\_name* argument in an LVS rule means that the particular rule pertains only to cells in that list, and that it may override other similar rules in those cells.

### Use with Trace Property

Conceptually, for cells with cell-specific LVS rules, a list of all relevant rules is maintained per cell. All non-cell-specific rules (basic rules) are entered in the list first. All cell-specific rule cancellations are done next in the list. All cell-specific rule additions are done last. For details on how rule cancellations are done, refer to [Trace Property](#).

### Notes and Restrictions

**Duplicate statements** — All usual restrictions that govern how many times a statement may appear in the rule file also govern how many times a statement may appear in a cell-specific list of statements. For example, the following is not allowed, because for cell list A there is more than one rule involving component type MP, subtype P, and source property name W:

```
LVS CELL LIST A cell1 cell2
TRACE PROPERTY MP(P) W W 2 CELL LIST A
TRACE PROPERTY MP(P) W W 3 CELL LIST A
```

**Duplicate cells** — A cell name may not appear in more than one LVS Cell List statement. For example, the following is not allowed, because the name cell2 appears in both lists:

```
LVS CELL LIST A cell1 cell2
LVS CELL LIST B cell2 cell3
```

When wildcards are used, this restriction becomes design-specific. The same cell can end up in two lists after wildcard substitution, even if the statements do not contain any identical names. For example, if cell list L1 contains the pattern A\* and cell list L2 contains the pattern \*B, then the cell AB belongs to both lists. If such a cell is detected, circuit comparison terminates with the NOT COMPARED status and the following error message is reported in the transcript:

```
ERROR: Duplicate cell name "AB" in cell lists "L1" and "L2" after wildcard
substitution.
```

**Hcell requirement** — Cells that appear in LVS Cell List statements, and are present in the source design, must serve as hcells. This restriction ensures that cell-specific rules are checked as intended, and that checks are not missed due to cell expansion.

When this restriction is violated, a secondary comparison status appears in the OVERALL COMPARISON RESULTS section of the LVS report, as follows:

```
Error: Non-hcells with cell specific rules.
```

The cells in question appear in a special section of the LVS report, together with the respective cell list names, as follows:

```
*****
NON-HCELLS WITH CELL SPECIFIC RULES
*****
(Cells that appear in LVS CELL LIST statements but do not serve as
hcells).
LVS CELL LIST clist_1_3 cell1 cel2 cel3
LVS CELL LIST clist_5_6 cel5 cel6
```

Note that cells not appearing in the design may appear in LVS Cell List statements even though they do not serve as hcells. This allows you to write LVS Cell List statements that are independent of a particular design or block. Recall that hcells may be established dynamically, for example, with the -automatch command line option (remember, the -automatch is not recommended unless the layout cells have the same devices as the source subcircuits sharing the same name).

**Unbalanced cells** — Generally, unbalanced cell expansion is treated as an error when applied to cells that appear in LVS Cell List statements. (Unbalanced cell expansion selectively expands hcells that are instantiated different numbers of times in the source and layout. It is controlled with the [LVS Expand Unbalanced Cells](#) specification statement).

Consider a cell called A, contained in a cell called B. Assume that A is expanded into B by unbalanced cell expansion. The expansion of A into B is treated as an error if any of the following conditions holds:

- The expanded cell A and the target cell B are in different cell lists; or
- The expanded cell A is in a cell list and the target cell B is not; or
- The target cell B is in a cell list and the expanded cell A is not.

Otherwise, the expansion is allowed.

Unbalanced cell expansion that violates this restriction is allowed to proceed, but it is reported as an error in the LVS report. A secondary comparison status appears in the OVERALL COMPARISON RESULTS section and also in the section pertaining to the particular target cell (cell B in the previous discussion), as follows:

```
Error:      Unbalanced cell expansion with inconsistent cell-list
membership.
```

The expanded cell names appear in the report section pertaining to the target cell, in a special sub-section of the report, as follows:

```
*****
```

### UNBALANCED CELL EXPANSION WITH INCONSISTENT CELL-LIST MEMBERSHIP

```
*****
```

(The following cells were expanded into this cell because there were different numbers of them in the layout and source. However, each expanded cell fits one of the following cases: (a) the expanded cell and this cell are in different cell lists, or (b) the expanded cell is in a cell list and this cell is not, or (c) this cell is in a cell list and the expanded cell is not.)

```
cell1 cell2
```

This restriction ensures that errors are reported when there is possibility of cell-specific rules being missed.

Cell names are source names:

As mentioned previously, all *cell\_name* arguments refer to the source design. For example, if the hcell list contains the pair:

```
A B
```

then the name B, not A, should be used in LVS Cell List statements. LVS uses B's cell-specific rules when comparing layout cell A to source cell B.

If the name A were used in a LVS Cell List statement, it would be ignored, and the cell list would not include the pair (A B). (That is, unless the source design also contained a cell called A. In that case, the LVS Cell List statement would apply to the source cell A and any corresponding layout cells).

One-to-many relations:

One-to-many hcell relations are handled properly. Consider this example:

```
LVS CELL LIST clistb_1_2 B1 B2
LVS CELL LIST clistb_3 B3
HCELL A B1
HCELL A B2
HCELL A B3
```

Calibre nmLVS-H will use rules pertaining to B1, B2, or B3 when comparing layout cell A to source cells B1, B2, or B3 respectively.

**Use basic rules** — If all rules of a given kind are cell-specific, and there are no basic rules of that kind, then generic cells that do not belong to any cell list will have no rules of that kind (unless the rule has a default setting). For example, if all Trace Property rules have CELL LIST arguments, and a particular cell does not appear in any list, then no properties will be traced in that cell. For this reason, it is good practice to always specify basic rules, and use cell-specific rules only to override the basic rules, as needed.

## Examples

### Example 1

In this example, the hcells from the source design are specified in my\_list. The Trace Property statement overrides the basic Trace Property rule for the cells in my\_list.

```
HCELL cellA layoutA
HCELL cellB layoutB

TRACE PROPERTY MP(P) W W //basic rule

LVS CELL LIST my_list cellA cellB

TRACE PROPERTY MP(P) W W 2 CELL LIST my_list
```

### Example 2

In this example, gate recognition is turned off for the analog\_block cell, which is presumed to be in the source netlist.

```
LVS RECOGNIZE GATES ALL //default

LVS CELL LIST gates_off analog_block

LVS RECOGNIZE GATES NONE CELL LIST gates_off
```

## LVS Cell Supply

Specification statement

**LVS CELL SUPPLY { NO | YES }**

### Parameters

- **NO**

Keyword that specifies not to propagate power and ground net paths from lower-level ports. This is the default behavior if you do not specify the statement in your rule file.

- **YES**

Keyword that specifies port names on cells matching the [LVS Power Name](#) and [LVS Ground Name](#) settings are to propagate power and ground net paths through the hierarchy into connected nets.

### Description

Controls propagation of power and ground net paths from lower level ports. A path in this context is a connected sequence of nets that are either power or ground. When NO is specified, or the statement is not used in the rule file, power and ground net paths are not propagated from lower-level ports.

When YES is specified, port names that match the LVS Power Name and LVS Ground Name settings propagate power and ground paths up and down the hierarchy into any nets connected to the ports. Specifying YES can be useful when top-level nets do not contain power and ground net names, but lower-level cells do.

Propagation of paths upward through the hierarchy continues to the top level or until a power or ground conflict is detected. If a conflict is detected, the entire path, at all levels of hierarchy, is designated as neither power nor ground. A conflict can be caused by two different supply paths propagating upward through the hierarchy until they connect to the same net. A conflict can also be caused by a supply path propagating from a lower-level cell and colliding with a power or ground name in the top cell.

Propagation of the paths downward through the hierarchy is affected by the LVS Cell Supply statement in only one way: if NO is specified, only paths connected to top-level power and ground names are propagated down through the hierarchy. If YES is specified, top-level nets that receive their path status by upward propagation are treated in the same way as nets named only at the top level.

In both cases, if a conflict is detected when propagating paths down through the hierarchy, and the same net of a particular cell is connected to power in one placement and ground in another placement, the net inside the cell is designated as neither power nor ground. The nets above that cell, however, retain their power and ground status. Note that the upward propagation of paths due to LVS Cell Supply YES, and the conflict resolution previously described, occur before the downward propagation.

Consider the following subcircuit:

```
.SUBCKT TOP
X1 1 2 3 4 inv
X2 1 2 3 4 inv_no_voltage
X3 1 2 3 4 BLOCK
.ENDS

.SUBCKT BLOCK 1 2 3 4
X2 1 2 3 4 inv_no_voltage
.ENDS

** Voltages come from VSS and VCC ports here
.SUBCKT inv IN OUT VSS VCC
MA VCC IN OUT p
MB OUT IN VSS n
.ENDS

** Voltages are propagated into this subcircuit
.SUBCKT inv_no_voltage IN OUT 1 2
MA 2 IN OUT p
MB OUT IN 1 n
.ENDS
```

When LVS Cell Supply YES is set and VCC and VSS are LVS Power and LVS Ground names, respectively, the power and ground paths from .SUBCKT inv are propagated upward into TOP. Since there are no conflicts, top-level nets 3 and 4 become ground and power nets, respectively. Supply paths are then propagated down into inv\_no\_voltage and BLOCK cells. In the inv\_no\_voltage cell, ports 1 and 2 are marked as power and ground because each instance of inv\_no\_voltage has ports 1 and 2 connected to nets that are power and ground. These nets are marked through propagation of signals from the cell inv. Note that downward propagation only occurs when lower-level cells are connected consistently through the hierarchy.

Power and ground conflicts for each cell that are caused by propagation of port supply paths are reported as errors in the summary section of the LVS report:

```
Error:      Power and/or ground ports connected to opposite voltage nets
and/or ports in layout.
Error:      Power and/or ground ports connected to opposite voltage nets
and/or ports in source.
```

In the detailed section for these cells, the specific nets involved in the conflict are reported.

## Examples

```
// Propagate power and ground net paths up the hierarchy
// from lower-level ports
LVS CELL SUPPLY YES
```

# LVS Center Device Pins

Specification statement

## LVS CENTER DEVICE PINS {NO | YES}

### Parameters

- **NO**

Keyword that specifies arbitrary coordinate points are returned between a pin layer and device seed layer. This is the default.

- **YES**

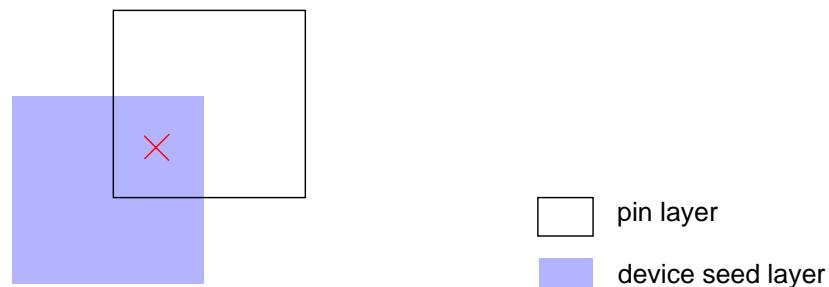
Keyword that returns a centered pin location coordinate between a pin layer and a device seed layer if a centered location can be identified.

### Description

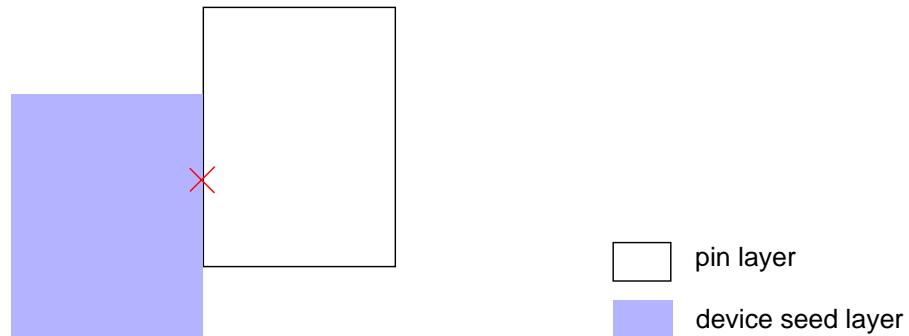
Pin locations available in the LVS Calibre Connectivity Interface flow represent an arbitrary point on the intersection of a pin layer and the device seed layer. For some extraction tools, greater accuracy is achieved if the pin is represented at the center of the intersection of the pin shape and the device seed shape. Specifying YES identifies a centered pin location if the intersection is rectangular or a single edge. Those centered coordinates are saved as the device pin location.

A centered pin location can be defined for a rectangular intersection area or along a single edge. For example, if the pin and seed shape overlap one another, the pin location is defined as the center of the rectangular intersection as seen in [Figure 4-155](#).

**Figure 4-155. Well-Defined Center Device Pin Location**

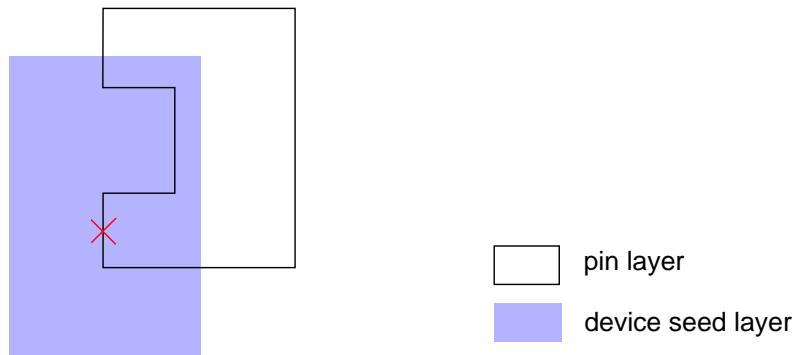


For a single edge boundary interaction, the middle of the edge intersection defines the centered pin location as seen in [Figure 4-156](#).

**Figure 4-156. Interaction Edge Pin Location**

marks the center pin location for the above pin and seed layer intersection

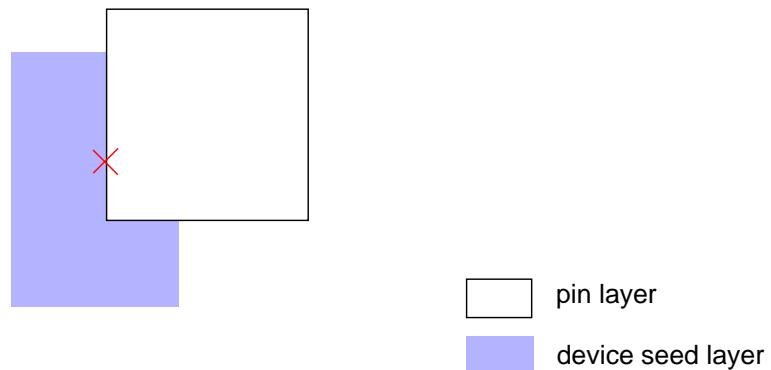
For irregular shapes where the center point is not well defined, an arbitrary location on the interaction boundary is presented as the pin location. For example, certain disjoint or bent interaction areas do not allow calculation of a centered pin location as seen in [Figure 4-157](#).

**Figure 4-157. Arbitrary Device Pin Location for Irregular Polygons**

marks an arbitrary pin location for the irregular pin layer and device seed layer interaction area

Similarly, the centered pin location is not available for multiple coincident edge interactions as seen in [Figure 4-158](#).

**Figure 4-158. Undefined Centered Pin Location**



✗ marks a possible center pin location when there are multiple coincident edge interactions

## Examples

```
//Use standard pin location reporting  
LVS CENTER DEVICE PINS NO
```

# LVS Check Port Names

Specification statement

## LVS CHECK PORT NAMES {NO | YES}

### Parameters

- **NO**

Keyword that instructs the tool *not* to compare names of matched top-level ports. This is the default behavior if you do not include this statement in your rule file.

- **YES**

Keyword that instructs the tool to compare the names of matched top-level ports and to report any naming conflicts.

### Description

Specifies whether the tool checks the names of matched ports. When you specify YES, the tool verifies that each top-level layout port name matches the corresponding source port name, and reports a discrepancy if no match is made.

When YES is specified, the tool reports a discrepancy in the Incorrect Ports section of the LVS report if any of the following are true:

- The layout port and the source port have user-given names, and the names are different.
- The layout port has a user-given name and the source port is unnamed.
- The layout port is unnamed and the source port has a user-given name.

The tool does not report a discrepancy if the following is true:

- The matched layout port and source port are unnamed, such as not having user-given names.

If [LVS Ignore Ports](#) YES is specified, then port names are *not* checked, even if LVS Check Port Names YES is specified.

See also [LVS Report Option NO](#) and [LVS Non User Name](#).

### Examples

#### Example 1

Default:

```
LVS CHECK PORT NAMES NO // top-level port names are not compared
```

#### Example 2

The LVS Ignore Ports statement affects whether top-level port names are checked:

```
LVS CHECK PORT NAMES YES
LVS IGNORE PORTS YES      // disables port name comparision
```

### Example 3

Example output for LVS Check Port Names YES:

```
*****  
INCORRECT PORTS  
*****  
DISC#      LAYOUT NAME          ne SOURCE NAME  
*****  
1          BB on net: BB        B on net: B  
2          3 on net: 3          C on net: C  
3          D on net: D          4 on net: 4
```

# LVS Compare Case

Specification statement

**LVS COMPARE CASE {NO | YES} {[NAMES] [TYPES] [SUBTYPES] [VALUES]}**

## Summary

Controls case sensitivity during LVS comparison.

## Parameters

- **NO**  
Keyword that specifies that all comparisons are case-insensitive.
- **YES**  
Keyword that specifies that all comparisons are case-sensitive (except as noted under NAMES). Specifying YES is equivalent to specifying NAMES TYPES SUBTYPES VALUES.
- **NAMES**  
An optional keyword specifying that net, instance, and port name comparisons (except port names of primitive built-in devices) are case-sensitive.  
  
Built-in (primitive) device names are never case-sensitive. Built-in device names for comparison are: M, MP, MN, ME, MD, LDD, LDDP, LDDN, LDDE, LDDD, J, L, D, C, R, Q, and V.  
  
[LVS Box](#) cells having names of built-in devices are treated as primitive devices for case comparison. Cells that are not empty and are not LVS Box cells are not treated as primitive devices, even when their names are built-in device names.  
  
Note that property names are always case-insensitive.
- **TYPES**  
An optional keyword that specifies that component type comparisons are case-sensitive. This includes cell names and user-defined device names but not built-in device names specified under NAMES.
- **SUBTYPES**  
An optional keyword that specifies that component subtype comparisons are case-sensitive.
- **VALUES**  
An optional keyword specifies that string property values should be treated as case-sensitive. This affects string value comparison in device reduction programs (effective property computation in the LVS Reduce family of statements using TOLERANCE STRING settings) and [Trace Property](#) STRING statements.

### Description

This statement controls case sensitivity during LVS circuit comparison. This statement controls identification of initial correspondence points, classification of objects by component type and subtype, classification of nets as power or ground, cell correspondence in hierarchical circuit comparison, processing of hcell lists in hierarchical circuit comparison, and other applications where text case is considered.

#### Note



If you use case-sensitive comparison, you should include the specification statements [Layout Case YES](#) and [Source Case YES](#). See “[Case-Sensitive Handling of Netlists](#)” in the *Calibre Verification User’s Manual*.

---

In nmLVS-H, the hcell list is case-insensitive by default but becomes case-sensitive if you specify LVS Compare Case YES or LVS Compare Case TYPES in the rule file. This statement also controls the processing of hcell lists in hierarchical netlist extraction (to ensure consistency with circuit comparison); otherwise it does not affect netlist extraction.

When LVS Compare Case NAMES or YES is specified, then LVS-related specification statements that involve net, instance, or port names are also case-sensitive. These include the following:

- [LVS Power Name](#)
- [LVS Ground Name](#)
- [Pathchk](#)
- [ERC Pathchk](#)

For example, when LVS Compare Case NAMES is requested, then

```
LVS Power Name Vcc vcc
```

specifies two different power names Vcc and vcc, and the name VCC is not a power name.

Similarly, when LVS Compare Case TYPES, SUBTYPES, or YES is specified, then LVS-related specification statements that involve component types or subtypes, respectively, are also case sensitive. These include the following:

- [Hcell \(TYPES\)](#)  
Also applies to an hcell list specified using the -hcell command line argument or cellnames matched using the -automatch command line argument.
- [Trace Property](#)
- [LVS Filter](#)
- [LVS Property Map](#)

When case-sensitive SUBTYPES are requested, then this statement:

```
LVS Filter R(x) open
```

filters out resistors with lowercase subtype x only, and this statement

```
LVS Filter R(X) open
```

filters out resistors with uppercase subtype X only.

### **Case comparison considerations —**

- See “[Selection of Extracted Netlist Names](#)” in the *Calibre Verification User’s Manual* for details on the hierarchical SPICE netlisting behavior.  
Case preservation during netlist extraction is not subject to LVS Compare Case.
- Flat LVS preserves the cases of element names, model names, and pin names from rule file Device statements when LVS reads in-memory extracted layout connectivity into the circuit comparison stage. LVS Compare Case controls case-sensitivity during LVS comparison.
- The [LVS Downcase Device](#) statement causes element names, model names, and pin names from rule file Device statements to be downcased. This statement provides backward-compatibility to deprecated functionality. Its use is discouraged.

## **Examples**

### **Example 1**

```
// Use case-insensitive comparisons  
  
LVS COMPARE CASE NO
```

### **Example 2**

```
// Make net, instance, and port name comparisons case sensitive.  
  
SOURCE CASE YES  
LAYOUT CASE YES  
LVS COMPARE CASE NAMES
```

### **Example 3**

```
// Handle source and layout in a case-sensitive manner.  
// Most netlist elements that differ by case are treated as  
// separate elements and compared as separate elements.  
  
SOURCE CASE YES  
LAYOUT CASE YES  
LVS COMPARE CASE YES
```

## LVS Component Subtype Property

Specification statement

### LVS COMPONENT SUBTYPE PROPERTY *name*

Used in ICtrace only.

#### Parameters

- *name*

A required name of a Design Architect component subtype property.

#### Description

Specifies the property name to be used in determining component subtypes for source instances. This statement can be specified once in your rule file. Sets the value of application variable lvs\_component\_subtype\_property.

#### Examples

```
// ICtrace uses subtype property called "model" for source instances
LVS COMPONENT SUBTYPE PROPERTY model
```

# LVS Component Type Property

Specification statement

**LVS COMPONENT TYPE PROPERTY** *name* [*name* ...]

Used in ICtrace only.

## Parameters

- *name*

A required name of a Design Architect component property. You can specify multiple names. The *name* can be a string variable.

## Description

Specifies a prioritized list of zero or more property names to be used in determining component types for source instances. This statement can be specified once in your rule file. Sets the value of application variable lvs\_component\_type\_property.

## Examples

```
// ICtrace uses component type property called "element"  
// for source instances  
LVS COMPONENT TYPE PROPERTY element
```

## LVS Cpoint

Specification statement

### LVS CPOINT *layout\_net\_name source\_net\_name*

#### Parameters

- *layout\_net\_name*

A required name of a net in the database specified in the [Layout Path](#) specification statement.

- *source\_net\_name*

A required name of a net in the database specified in the [Source Path](#) specification statement.

#### Description

Specifies a correspondence point between a layout net and a source net. With this information, an LVS application can match the layout and source databases through the use of the specified net names. Correspondence points specified in this manner are referred to as *cpoints*.

Net names in this statement refer to the top level cell, which is specified in the [Layout Primary](#) and [Source Primary](#) specification statements. All hierarchical pathnames are rooted in the top level cell.

The format of net names for the LVS Cpoint statement depends on the [Layout System](#) and [Source System](#) specification statements and generally follow the format used for reporting nets in the LVS report. These formats are described in the section “Net and Instance Names” of the [Calibre Verification User’s Manual](#). However, for cpoints there are the following exceptions:

- When referring to SPICE netlists, you should use the Calibre nmLVS-H naming convention. Specifically, all subcircuit call names should include the subcircuit call designator X, both in flat and hierarchical LVS. For example, a hierarchical pathname should be written as “X1/X2/X3/N1” and not as “1/2/3/N1”.
- When referring to Mask-mode layout, such as in Mask-mode ICtrace, or flat Calibre nmLVS with a geometric Layout System, net names should appear without the (x,y) location suffix that is usually printed in the LVS report. For example, a net name should be written as ABC and not ABC(30,5).

Net names that contain special characters, such as hierarchical delimiters (/), must be enclosed in quotation marks. It is good practice to enclose all names in quotation marks.

Following is a description of the net name formats used in various systems:

**SPICE** — A net in a SPICE netlist is identified by its hierarchical pathname. A hierarchical pathname has the form:

/<subckt-call-name-1>/.../<subckt-call-name-n>/<node-name>

where “n” is zero or more. Subcircuit call names should include the SPICE subcircuit call designator X, both in hierarchical and flat LVS. Note that this is different from the naming

convention used for reporting nets in the LVS report in flat LVS, where the X designator is omitted. Examples:

```
NETA
15
"X1/X2/NETB"
"XABC/X4/X7/16"
```

In flat LVS, nets in SPICE netlists must be identified by their highest-level component in the SPICE hierarchy. In the following example:

```
.SUBCKT AAA 1 2
C1 1 2
.ENDS
X1 3 4
```

the name “3” would be valid in flat LVS, but the name “X1/1” would not be found because net X1/1 is connected to a pin of subcircuit call X1.

**Mask layout** — A texted layout net in Mask LVS is identified by its name. Untexted layout nets in Mask LVS cannot form cpoints. All names may be used, including names that contain “/” characters. All characters in a name are used; leading “/” characters are *not* ignored. Mask LVS includes flat Calibre nmLVS with [Layout System GDSII](#) (or other geometric layout) as well as ICtrace(M). Example:

```
NETA
```

**Direct layout** — A named Pyxis Layout net in Direct LVS is identified by its name. Unnamed layout nets in Direct LVS cannot form cpoints. All names may be used, including names that contain “/” characters. All characters in a name are used; leading “/” characters are *not* ignored. Example:

```
NETA
```

**EDDM viewpoints in ICtrace** — A net in an EDDM database is identified by its hierarchical pathname. A hierarchical pathname has one of two forms:

```
/<instance-name-1>/.../<instance-name-n>/<net-name>
//<global-net-name>
```

where “n” is zero or more. Net names are either user given names, or system generated names in the form n\$<number>. Examples:

```
" /NETA "
"/n$23"
"/i$7/i$8/ABC"
"/i$7/i$8/n$23"
" //VCC "
```

**Cnet** — Only nets with user given names can form cpoints in Cnet databases. User given is defined according to the conventions of the original system from which the Cnet database was created. Nets are identified by their user-given names. Example:

```
NETA
```

### Internal Identification of Cpoints

To process and identify cpoints, the LVS circuit comparison engine generates names of the form:

```
icv_cpoint_x
```

where x is an integer, and assigns them internally to nets that form cpoints. These generated names are treated in a similar way to original database names. They are not used for reporting discrepancies, but they may appear in other parts of the LVS report, for example in lists of Conflicting Layout Names and Conflicting Source Names.

### Effect Upon Circuit Transformations

Cpoints affect circuit transformations much like initial correspondence points do. In particular, nets specified in LVS Cpoint statements cannot be removed by unused device filtering or series reduction, and cannot be made internal to LVS-generated structures such as logic gates or injected components. For example, if a cpoint is attached to a net internal to a logic gate, the gate will not be formed. This ensures that all cpoints are preserved and no cpoints are lost.

### Unused and Erroneous Cpoints

LVS ignores cpoints that involve nets discarded during circuit comparison. For example, a cpoint is ignored if it specifies a net connected only to an unused device. LVS also ignores cpoints that refer to names that do not exist in the layout or source. In Calibre, warnings appear in the transcript. For example:

```
WARNING: lvs cpoint name 1/5 not found in source.
```

You cannot use cpoints to specify correspondence between one layout net and several different source nets, or between one source net and several different layout nets. In other words, you can not use cpoints to connect together different nets. For example, the combination of statements:

```
lvs cpoint "A" "C"  
lvs cpoint "A" "D"
```

do not connect together nets C and D in the source. These two cpoints are not used and instead are reported as Conflicting Layout Names in the LVS report (using their icv\_cpoint\_x generated names).

Similarly, you cannot use cpoints to override regular initial correspondence points, such as elements with the same name in the layout and source. For example, if the layout and source both contain a net called “A”, then you cannot specify the statement:

```
lvs cpoint "A" "C"
```

because that conflicts with the initial correspondence point “A”. If you do this, the tool does not use either the cpoint nor the initial correspondence point. Instead, they are reported as Conflicting Layout Names in the LVS report.

All cpoints that cannot be used by LVS are reported in the Information and Warnings section of the LVS report.

## Cpoint Information in the Report and Transcript

The tool lists all cpoints used in the Information and Warnings section of the LVS report. For example:

- o Cpoints:

Layout	Source
-----	-----
1/3	1/2/7
1/4	1/2/8

Cpoints that cannot be used by LVS are also listed in the Information and Warnings section of the LVS report. For example:

- o Failed Cpoints:

Layout	Source
-----	-----
1/5	1/2/9
1/6	1/2/10

Note that the format used for reporting cpoint net names in the LVS report and transcript is the same as for nets in general. This may be different from the way cpoints are entered in the rule file. Specifically, flat LVS omits the X subcircuit call designators in hierarchical pathnames in SPICE.

Note also that when cpoints appear in lists of Conflicting Layout Names and Conflicting Source Names in the LVS report, they are identified by their generated names in the form `icv_cpoint_x`, not by their original names.

All cpoints (both used and unused) appear in the Calibre nmLVS transcript under the header `LVS CPOINTS`. In addition, the Calibre nmLVS transcript shows the internally-generated `icv_cpoint_x` names assigned to cpoint nets:

```
PROCESSING CPOINTS...
Attaching name icv_cpoint_0 to net 1/3 in layout and net 1/3 in source
```

## Examples

```
/* Treat the following as initial correspondence points in layout and
source. */
LVS CPOINT "AAA" "X1/X2/5"
LVS CPOINT "BBB" "CCC"
LVS CPOINT "DDD" "7"
LVS CPOINT "X3/X4/5" "X6/N1"
```

# LVS Device Type

Specification statement

**LVS DEVICE TYPE** *built\_in\_type user\_type* [*user\_type ...*]  
 [‘[’ *built\_in\_pin\_name=user\_pin\_name*  
     *built\_in\_pin\_name=user\_pin\_name*  
     *[built\_in\_pin\_name=user\_pin\_name ...]’’*]  
[SOURCE] [LAYOUT]

## Summary

Allows specification of device instances having arbitrary device names and pin names to serve as built-in device instances in ERC and LVS comparison.

## Parameters

- *built\_in\_type*

A required built-in [Device](#) type. This parameter is not case-sensitive. Table 4-21 shows the allowed types:

**Table 4-21. Device Types**

<i>built_in_type</i>	Component Name	Description
NMOS	MN	CMOS N transistor
PMOS	MP	CMOS P transistor
ENH or ENHANCEMENT	ME	NMOS enhancement transistor
DEPL or DEPLETION	MD	NMOS depletion transistor
MOS	M	MOS generic transistor
LDNNMOS	LDDN	CMOS lightly doped drain N transistor
LDPPMOS	LDDP	CMOS lightly doped drain P transistor
LDDENH or LDDENHANCEMENT	LDDE	NMOS lightly doped drain enhancement transistor
LDDEPL or LDDEPLETION	LDDD	NMOS lightly doped drain depletion transistor
LDDMOS	LD	MOS lightly doped drain generic transistor
BIPOLAR	Q	bipolar transistor
CAPACITOR	C	capacitor
DIODE	D	diode
RESISTOR	R	resistor

- ***user\_type***

A required (arbitrary) name of a user-defined device component type. You may specify this parameter multiple times in a single statement. Case-sensitivity follows what is used in the corresponding Device statement in the rule file.

- **[’ *built\_in\_pin\_name=user\_pin\_name built\_in\_pin\_name=user\_pin\_name ... ’]***

Optional set of parameters, enclosed in square brackets, which specify the pin mapping of user-given pin names to built-in device pin names. If the pin mapping section is omitted, then the user-defined device must already use the expected built-in pin names. The order of the pin names is arbitrary; however, if the mapping for one of the pins is specified, the other pin(s) must be mapped as well. Pin names are not case-sensitive. The sets of built-in pin names that must be mapped for each device type are shown in this table.

**Table 4-22. Built-In Device Pin Names for Mapping**

Built-In Type	Pin Names
BIPOLAR	C, B, E
CAPACITOR	POS, NEG
DIODE	POS, NEG
MOS types	S, G, D
RESISTOR	POS, NEG

- **SOURCE**

An optional keyword that specifies the statement applies to the source design. This option is used by default.

- **LAYOUT**

An optional keyword that specifies the statement applies to the layout design. This option is used by default.

### Description

Specifies that instances appearing in the layout and source with the indicated ***user\_type*** should be classified as devices of the indicated ***built\_in\_type*** during LVS circuit comparison. This statement has *no effect* upon layout device extraction. This statement also applies to ERC path checking as described later in this section. LVS Device Type statements having the same ***user\_type*** in the rule file cause a compiler error.

This designation means that instances of the indicated ***built\_in\_type*** participate in special processing that built-in devices receive, for example, logic gate recognition, unused device filtering, series and parallel reduction, and so forth. For a complete list of these special processing features refer to the “Built-In Device Types” section in the [Calibre Verification User’s Manual](#).

Suppose there is this statement in the rule file:

```
LVS DEVICE TYPE NMOS NTRAN7
```

and this in the source netlist:

```
* Source:  
.SUBCKT NTRAN7 g s d b  
.ENDS  
...  
X0 3 4 GND GND NTRAN7  
...
```

In this example, instances with component type NTRAN7 participate just like MN devices in the formation of CMOS logic gates (as controlled by [LVS Recognize Gates](#)), in unused device filtering (as controlled by [LVS Filter Unused Option](#)), and in parallel transistor reduction (as controlled by [LVS Reduce Parallel MOS YES](#)). Note that the participating instances retain their original component types; that is, the component names are not replaced. This behavior occurs so long as the NTRAN7 devices use the *required* built-in pin names that LVS expects. User-defined pin names can be remapped as necessary using the pin-mapping list.

Devices are always matched by component type (and model name, when applicable) in the source and layout and not by the *built\_in\_type* in the rule file. For the preceding example, this means NTRAN7 devices do not match built-in MN devices. The NTRAN7 devices participate in all of the special processing received by built-in MOS devices, but they will only match other devices of *user\_type* NTRAN7. This means LVS Device Type component types are *not* subject to statements where the component type is explicitly indicated and it is different from NTRAN7. In this case, NTRAN7 instances are not subject to an [LVS Reduce](#) MN Parallel statement or to a [Trace Property](#) MN W W 0 statement; those statements apply only to instances where the component type is actually MN.

Pin names of participating instances must obey the same restrictions as for the respective built-in devices for LVS comparison; otherwise, the instance is classified as a non-standard device and does not function as a device of the indicated built-in type. This is shown in the LVS report with a “\*\* non-standard device \*\*” note next to the component type in the NUMBERS OF OBJECTS sections. For details about pin naming, refer to “Built-In Device Types” section in the [Calibre Verification User’s Manual](#).

The pin-mapping specification [ *built\_in\_pin\_name=user\_pin\_name...* ], which appears in square brackets, allows user-defined pin names to correspond to built-in pin names.

Note that the LVS circuit comparison module is less restrictive about certain pin-naming conventions than the Device statement syntax. For example, the Device statement syntax requires that built-in MOS devices use “B” for the fourth (bulk) pin. However, LVS circuit comparison does not enforce this convention, so user-defined devices specified as a MOS-family *user\_type* may use any name for the fourth pin and the corresponding devices behave as built-in. For example, suppose you have this in your source netlist:

```
* Source:  
.SUBCKT arb g s d z  
.ENDS  
...  
X0 3 4 PWR PWR arb  
...
```

and you have this Device statement in your rule file:

```
device arb seed seed(g) sd(s) sd(d) well(z)
```

Then arb devices in the layout are extracted to match the source netlist. If you then specify:

```
LVS DEVICE TYPE PMOS arb
```

in your rule file, then arb devices are matched together and receive the same treatment as built-in MOS devices, even though the bulk pin name “z” of the arb device does not match what is ordinarily required in a Device statement for MOS devices. The G, S, and D pin names are required, as for all built-in MOS devices. Again, “arb” devices are only matched to other arb devices and not to any other component types.

This statement operates on instances of primitive components, including instances of [LVS Box](#) cells, but does not operate on instances of non-primitive hcells (that is, hcells that, in turn, contain other instances).

## Restrictions and Consistency Checking

If the layout defined by the [Layout System](#) statement contains devices, and the rule file contains Device statements for the user-defined types being mapped (these conditions presume connectivity extraction is being performed during the run), then the list of mapped pin names is validated at compile time. The following checks are performed:

- If any user pin names are specified in an LVS Device Type specification statement but are not present on the device, then an error is issued at rule compilation time:

```
DEVICE specified in LVS DEVICE TYPE statement missing required pin
```

- If no user-defined pin names are specified in an LVS Device Type specification statement, then the compiler verifies that the corresponding Device statement contains the built-in pin names. If the Device statement is missing any required pins, then a warning is issued. In this case, LVS classifies the device as a non-standard device and a warning is issued in the transcript at rule compilation time:

```
DEVICE specified in LVS DEVICE TYPE statement missing builtin pin
```

A device can be given different pin mappings in separate LVS Device Type statements specifying the LAYOUT and SOURCE keywords, where the statements reference the same device. The source and layout devices will match during comparison as long as the *user\_type* and *built\_in\_type* are the same. For example, if resistors of type RDEV in the layout have the pins PLUS and MINUS, but resistors of type RDEV in the source have pins A and B, then the following LVS Device Type specification statement allows these devices to match during comparison:

```
LVS DEVICE TYPE RESISTOR RDEV [ POS=PLUS NEG=MINUS ] LAYOUT
LVS DEVICE TYPE RESISTOR RDEV [ POS=A NEG=B ] SOURCE
```

If both the *user\_type* and the *built\_in\_type* are not identical, then devices are not matched during comparison.

The [LVS Filter](#) ... SHORT and [LVS Reduce](#) specification statements do not support the specification of mapped user pins names in LVS Device Type statements. The original (user-given) pin names must be used in these specification statements.

### Usage in ERC Path Checking

ERC path checking checks paths through the pins of built-in MOS and resistor devices by default. You can use LVS Device Type statements with pin mapping to include user-defined devices during path checking. To add bipolar devices, capacitors, or diodes, you must also specify the appropriate [ERC Path Also](#) statement. Only LVS Device Type statements that have the LAYOUT keyword (which is part of the default behavior) apply to ERC. See “[User-Defined Devices in Path Check Operations](#)” in the *Calibre Verification User’s Manual*.

### Logic Gate Naming

Some types of logic gate structures normally contain the component type as part of the name of the gate. When such gate structures are formed from component types that are subject to the LVS Device Type specification statement, the component type appears in the name of the gate in parentheses. For example, if you specify:

```
LVS DEVICE TYPE PMOS UDP
```

then LVS may form gates of the following types:

- S(UDP)n—similar to SMPn
- SP(UDP)\_n1\_n2\_...\_nm—similar to SPMP\_n1\_n2\_...\_nm
- SP(UDP)(*expression*)—similar to SPMP(*expression*)

where n, n1, n2, nm are integer numbers and *expression* is described in the section “Logic Gate Recognition” of the *Calibre Verification User’s Manual*.

### Examples

#### Example 1

This example specifies that instances with component type NTRAN7 should serve as NMOS transistors in circuit comparison in layout and source. NTRAN7 devices participate just like MN devices in the formation of CMOS logic gates, in unused device filtering, and so forth, but are not matched to MN devices.

```
LVS DEVICE TYPE NMOS NTRAN7
```

#### Example 2

This example specifies that instances with component type RDEV should serve as built-in resistor devices in the layout during circuit comparison. The pin names PLUS and MINUS are mapped to the built-in names POS and NEG. RDEV devices will also participate in ERC path checking.

```
LVS DEVICE TYPE RESISTOR RDEV [ POS=PLUS NEG=MINUS ] LAYOUT
```

**Example 3**

This example specifies that instances with component type PCAP should serve as built-in capacitor devices during circuit comparison. Pin names for PCAP instances are expected to conform to built-in pin-naming conventions POS and NEG. PCAP devices will also participate in ERC path checking.

```
DEVICE PCAP (ptype) poly_cap cap_pin(POS) cap_pin(NEG) [ ... ]
LVS DEVICE TYPE CAPACITOR PCAP // allow PCAP to be treated like a
                                // built-in C device
ERC PATH ALSO CAPACITOR      // include capacitors in ERC path checks
```

## LVS Discard Pins By Device

Specification statement

### LVS DISCARD PINS BY DEVICE {NO | YES}

#### Parameters

- **NO**

Keyword that specifies not to discard extra pins from input devices during circuit comparison. This is the default behavior.

- **YES**

Keyword that specifies to discard extra pins from input devices during circuit comparison.

#### Description

Specifies LVS should discard extra pins from recognized devices during circuit comparison. This statement may appear once.

Pins are discarded from input devices during circuit comparison as follows.

For every primitive instance (in both layout and source):

1. LVS looks in the rule file for a **Device** operation with the same component type (*element\_name*), the same component subtype (*model\_name*), the same number of pins, and the same pin names as in the instance. If such a Device operation exists then no pins are discarded.
2. Else, LVS looks in the rule file for a Device operation with the same component type (*element\_name*) and the same component subtype (*model\_name*) as in the instance, and having pin names that are a subset of the pin names in the instance. If there is a *unique* such Device operation, then all instance pins not in the Device operation are discarded. Unique here means that there is *at least* one such Device operation and, if there is more than one, then they all have the same number of pins, same pin names, and same pin swappability conditions.
3. Otherwise, no pins are discarded from the instance.

Notes:

- If the instance has no component subtype (*model\_name*), then LVS looks for a Device operation with no model name.
- If the input database has inherent pin swappability information (specifically, ICtrace(D) layout, or CNET derived from ICtrace(D) layout, or from ICtrace(M) layout, or from Calibre geometric database systems) then, to achieve a match, the Device operation pin swappability must be compatible with the input instance pin swappability.
- All name comparisons are case-insensitive.

When you specify NO (the default), pin filtering, as previously described, does not occur. However, regardless of how this statement and rule file Device statements are specified, LVS

may discard pins during circuit comparison, when necessary, to correlate layout and source. Such occurrences are reported as errors or warnings in the LVS report.

This statement is useful, for example, when you have 4-pin MOS devices in the source but 3-pin MOS devices in the layout (when it may be difficult to determine bulk connectivity in the layout). Example 1 shows this.

One solution formerly used in such cases was to specify [LVS Power Name](#) B. This would allow LVS to return a CORRECT result by turning missing-pin errors into warnings. That solution is obsolete; use LVS Discard Pins By Device YES instead.

## Examples

In all examples, assume that LVS Discard Pins By Device YES is specified.

### Example 1

In this example, LVS discards the bulk pin (connected to net 4) from M1. This statement is useful when you have 4-pin MOS devices in the source but 3-pin MOS devices in the layout (for example, when it is difficult to determine bulk connectivity in the layout).

Rule file:

```
DEVICE MN(N) ngate poly nsd nsd // 3-pin MN(N)
```

SPICE:

```
M1 1 2 3 4 N $$ 4-pin MN(N)
```

### Example 2

In this example, LVS discards the bulk pin from M1 but not from M2. The decision is made by model name.

Rule file:

```
DEVICE MN(NA) nga poly nsd nsd // 3-pin MN(NA)
DEVICE MN(NB) ngb poly nsd psub // 4-pin MN(NB)
```

SPICE:

```
M1 1 2 3 4 NA $$ 4-pin MN(NA)
M2 5 6 7 8 NB $$ 4-pin MN(NB)
```

### Example 3

In this example, LVS discards the substrate pin (connected to net 3) from C1.

Rule file:

```
DEVICE C cap1 poly met1 // 2-pin C, no model
```

SPICE:

```
C1 1 2 C=100 $SUB=3 $$ 3-pin C, no model
```

### Example 4

In this example, LVS discards pins A and C (connected to nets 1 and 3 respectively) from X1.

Rule file:

```
DEVICE FRED seed bl(B) dl(D) // 2-pin FRED, no model
```

SPICE:

```
.SUBCKT FRED A B C D $$ 4-pin FRED, no model
.ENDS
X1 1 2 3 4 FRED
```

### Example 5

In this example, pins are not discarded from M1 because a matching 4-pin Device operation exists, as described in [Example 1](#) above. The 3-pin Device operation does not apply.

Rule file:

```
DEVICE MN(N) ng1 poly nsd nsd // 3-pin MN(N)
DEVICE MN(N) ng2 poly nsd pwell // 4-pin MN(N)
```

SPICE:

```
M1 1 2 3 4 N $$ 4-pin MN(N)
```

### Example 6

In this example, pins are not discarded from M1 because there is no Device operation with element name MN and model name NB.

Rule file:

```
DEVICE MN(NA) nga poly nsd nsd // 3-pin MN(NA)
```

SPICE:

```
M1 1 2 3 4 NB $$ 4-pin MN(NB)
```

### Example 7

In this example, pins are not discarded from C1 because there is no Device operation with element name C and model name A. The Device operation with unspecified model does not apply to C(A) devices.

Rule file:

```
DEVICE C cap1 poly met1 // 2-pin C, no model
```

SPICE:

```
C1 1 2 C=100 $SUB=3 $[A] $$ 3-pin C(A)
```

# LVS Downcase Device

Specification statement

## LVS DOWNCASE DEVICE {NO | YES}

### Parameters

- **NO**

Keyword that instructs LVS to use the same case of letters for element names, model names, and pin names as is used in [Device](#) statements. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Use of YES is discouraged. Keyword that instructs LVS to use lowercase for all element names, model names, and pin names found in Device statements.

### Description

This statement is provided to restore deprecated functionality in LVS. Under most circumstances, you should use the default behavior.

This statement restores prior behavior with respect to case handling in Device operations by instructing LVS to use lowercase representations of names. Specifically, a YES setting has the following effects:

- It instructs the hierarchical SPICE netlister to write the output netlist using lowercase representations of element names, model names, and pin names from rule file Device operations.
- It instructs flat LVS to use a lowercase representation of element names, model names, and pin names from rule file Device operations when LVS reads in-memory extracted layout connectivity into the circuit comparison stage.
- It instructs the hierarchical SPICE netlister to ignore case when checking for duplicate element names during generation of primitive .SUBCKT definitions for user-defined devices (because element names are downcased in the output netlist).
- It instructs the rule file compiler to use case-insensitive comparisons for element and model names when performing consistency checks of certain LVS specifications statements with respect to Device operations, regardless of the [LVS Compare Case](#) setting. The relevant specification statements are listed below; the relevant fields in the Device operation are indicated.

[Trace Property](#) — Element name, model name.

[LVS Property Map](#) — Element name.

[LVS Filter](#) — Element name, model name.

[LVS Split Gate Ratio](#) — Element name.

## LVS Downcase Device

---

A NO setting (the default) instructs LVS to preserve case of element, model, and pin names from Device operations as described previously. This statement may appear at most once.

### Examples

```
// use text case in the DEVICE statement for netlisting
LVS DOWNCASE DEVICE NO
```

## LVS EDDM Process M

Specification statement

### LVS EDDM PROCESS M {NO | YES}

Used only in ICtrace.

#### Parameters

- **NO**  
Keyword that instructs ICtrace *not* to apply M property values to certain other property values in EDDM databases. This is the default behavior.
- **YES**  
Keyword that instructs ICtrace to apply M property values to certain other property values in EDDM databases.

#### Description

This statement controls how ICtrace applies M properties to other properties in an EDDM database, in a manner that is similar to SPICE semantics. This statement may be specified at most once. This statement has no effect in Calibre tools.

When NO is specified, M properties are not used to modify other properties in an EDDM database.

#### Regular EDDM Properties

This section applies when YES is specified.

The values of certain properties in some devices in EDDM databases are adjusted based on the values of optional M properties owned by those devices. These adjustments emulate the semantics of M parameters in SPICE netlists.

Specifically, if LVS EDDM Process M YES is specified, and an instance in a EDDM database has a property called M with a numeric value, and the LVS component type of the instance is one of those listed in [Table 4-23](#), then property values are multiplied by the M value as described in the table.

**Table 4-23. EDDM Property Processing**

LVS Component Type	Properties multiplied by M value
C	C, W
D	AREA, PJ
J	AREA, W
L	L, R

**Table 4-23. EDDM Property Processing (cont.)**

LVS Component Type	Properties multiplied by M value
M, MN, MP, ME, MD, LDD, LDDN, LDDP, LDDE, LDDD	W, AS, AD, PS, PD
Q	AREA, W
R	R, W

In addition, when an EDDM instance is classified as a built-in type by means of the [LVS Device Type](#) specification statement, then LVS EDDM Process M YES applies to the instance as if its LVS component type were the respective built-in component type.

For example, if the LVS component type of the instance is mydev, and the following statement is specified in the rule file:

```
LVS DEVICE TYPE NMOS mydev
```

then LVS EDDM Process M YES applies to the instance as if its LVS component type were MN.

Notes:

- The [LVS Spice Replicate Devices](#) and [LVS Spice Multiplier Name](#) specification statements have no effect in EDDM databases.
- The LVS component type of an EDDM instance is determined according to [LVS Component Type Property](#) properties and other data, as described under “[Component Types](#)” in the *Calibre Verification User’s Manual*. Additionally, the [LVS Device Type](#) specification statement may be applied, as described previously.
- Properties are adjusted only in EDDM instances that ICtrace directly operates on. Typically, those instances are designated as “primitive” in the EDDM viewpoint. Property adjustment occurs only when ICtrace directly accesses the property, for example as part of a [Trace Property](#) statement.
- Property adjustment is internal to ICtrace and does not affect EDDM database content, nor the evaluation of expressions in the EDDM database. For example, if an EDDM instance has the following properties:

```
W = 2
M = 3
Z = W + 1  (property of type "expression")
```

and the LVS component type of the instance is M, and LVS EDDM Process M YES is specified, then, in ICtrace, property W receives the value  $2 * 3 = 6$ , but property Z receives the value  $2 + 1 = 3$ . In other words, the original value of W is used to evaluate the EDDM expression for Z.

- All property names and all LVS component type names are case-insensitive. For example, the LVS EDDM Process M specification statement applies equally to property names W and w; AREA, area, and ARea. In the same way, it applies to component types like Q and q, and so forth.
- If the value of the M property is not numeric, then the M property is silently ignored.

### SPICE-Like Property Syntax

The “Spice-Like Property Syntax” section of the *ICverify User’s and Reference Manual* discusses details of property specification in EDDM and Pyxis databases. Refer to that section for a discussion of allowed syntax and semantics.

### Examples

Assume LVS EDDM Process M YES is specified. Also assume there is an MP instance that has the following properties:

```
M=2  
W=3 . 2E-9
```

Then ICtrace calculates W to be 6.4E-9 when processing that instance. The original EDDM database is not affected.

## LVS Exact Subtypes

Specification statement

### LVS EXACT SUBTYPES {NO | YES}

#### Parameters

- **NO**

Keyword that instructs LVS *not* to use the device subtype exact matching algorithm. This is the default behavior.

- **YES**

Keyword that instructs LVS to match device instances with different component subtypes independently of ambiguity resolution.

#### Description

This statement controls how circuit comparison matches devices with unspecified component subtypes with devices that have specified component subtypes.

If both LVS Exact Subtypes NO and [LVS Strict Subtypes](#) NO are specified, then a device with no subtype specified can be matched to a corresponding device of the same type and *any* subtype, empty or non-empty, without an error. In this case, devices that use subtypes are first matched without using subtypes. If this initial match is ambiguous, then subtypes are used to resolve such ambiguities.

If LVS Exact Subtypes NO and LVS Strict Subtypes YES are specified, then strict subtype matching is performed.

If LVS Exact Subtypes YES is specified, device subtypes are used together with device types when devices are initially matched. Empty subtypes match only empty subtypes. The additional step of resolving ambiguities is not necessary.

Note that matching exact subtypes forces LVS Strict Subtypes YES, no matter what the LVS Strict Subtypes setting is. The difference between exact subtype matching and strict subtype matching is exact matching checks the devices independent of any ambiguity resolution. Strict subtype checking is performed after ambiguity resolution. As a result, exact subtype matching can report connectivity errors that strict matching does not.

This statement may only be specified once.

## Examples

Given these two netlists:

```
Layout:      R1 1 2 100          $$ No subtype specified.  
Source:      R1 1 2 100 $[RTYPE1]  $$ Subtype RTYPE1.
```

The rule file statement LVS Exact Subtypes YES reports the following discrepancy:

```
-----  
21    R1   R           R1   R(RTYPE1)  
      bad component subtype  
-----
```

In addition, connectivity errors like mismatched nets can appear because of the subtype mismatch.

## LVS Exclude Hcell

Specification statement

**LVS EXCLUDE HCELL *cell\_name* [*cell\_name* ...]**

Used only in Calibre nmLVS-H and PERC.

### Parameters

- *cell\_name*

A required cell name that instructs Calibre nmLVS-H and the Query Server to prevent that cell from serving as an hcell. A list of cell names can be used. A cell parameter may contain one or more asterisk (\*) characters; the \* character is a wildcard that matches zero or more characters. A string that contains a \* character must be enclosed in quotation marks because the \* is a reserved symbol.

### Description

This statement prevents the specified cells from serving as hcells in nmLVS-H. Any cell specified in an LVS Exclude Hcell statement does not serve as an hcell under any circumstances. This affects hcells specified explicitly with the -hcell command line option, or with **Hcell** specification statements in the rule file, as well as hcells specified implicitly with the -automatch command line option. See “[Calibre nmLVS/nmLVS-H](#)” in the *Calibre Verification User’s Manual* for a discussion of nmLVS command line options.

This specification statement also affects the Query Server netlist analysis functionality. Excluded hcell names are never used as hcells in the NETLIST HCELL, NETLIST HCELLS, NETLIST AUTOMATCH and NETLIST PLACEMENTMATCH commands, or in rule file Hcell statements, when processed by the Query Server. See “[Netlist and Hierarchy Analysis Commands](#)” in the *Calibre Query Server Manual*.

### Examples

#### Example 1

This example prevents cells “aaa” and “bbb” from serving as hcells:

```
LVS EXCLUDE HCELL "aaa" "bbb"
```

#### Example 2

This example prevents all cells whose names start with the characters “bar” from serving as hcells:

```
LVS EXCLUDE HCELL "bar*"
```

# LVS Execute ERC

Specification statement

## **LVS EXECUTE ERC {YES | NO}**

### Parameters

- **YES**

Keyword that instructs LVS to perform selected ERC rule checks. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs Calibre not to perform ERC rule checks.

### Description

Specifies whether LVS performs ERC rule checks during circuit extraction. The layout database must be geometric to perform ERC. You can specify this statement once in your rule file.

ERC rule checks are performed according to the [ERC Select Check](#) and [ERC Unselect Check](#) settings in the rule file. Any DRC-style rule check may be specified in these statements; however, the [Pathchk](#) operation is the one most commonly used in an ERC rule check.

The following commands allow for ERC in Calibre nmLVS:

```
calibre -lvs -hier ... rule_file          // hierarchical when
                                            // Layout System is geometric
calibre -spice ... rule_file              // hierarchical
calibre -lvs rule_file                  // flat
calibre -lvs -tl [options] rule_file    // flat
```

ERC results are not immediately made part of the active cell in ICtrace. The ERC results database must be explicitly loaded into Pyxis Layout.

See the “Electrical Rule Checks” chapter in the [Calibre Verification User’s Manual](#) for more information on ERC.

You can specify ERC options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Running ERC Checks](#)” in the [Calibre Interactive and Calibre RVE User’s Manual](#).

This statement has no interaction with [ERC Pathchk](#).

### Examples

This example shows typical usage of ERC statements.

```
LVS EXECUTE ERC YES
ERC SELECT CHECK erc_checks
GROUP erc_checks erc_1 erc_2 erc_3
ERC RESULTS DATABASE erc.db

erc_1 {PATHCHK !POWER PRINT POLYGONS "no_power"}
erc_2 {PATHCHK !GROUND PRINT POLYGONS "no_ground"}
erc_3 {PATHCHK !LABELED}
```

## LVS Expand Seed Promotions

Specification statement

### LVS EXPAND SEED PROMOTIONS {NO | YES}

Used only in Calibre nmLVS-H and PERC.

#### Parameters

- **NO**  
Keyword that instructs nmLVS-H not to expand hcells that were subject to seed promotion during device recognition. This is the default behavior if you do not specify this statement.
- **YES**  
Keyword that instructs the nmLVS-H circuit comparison module to expand hcells that were subject to seed promotion during hierarchical device recognition.

#### Description

Instructs the hierarchical nmLVS circuit comparison module whether or not to expand hcells that were subject to seed promotion during device recognition. Seed promotion occurs when hierarchical device recognition is unable to completely process devices within a cell, and promotes those devices out of the cell and places them at a higher level of hierarchy. When YES is specified, cells that were subject to seed promotion out of the cell do not serve as hcells in hierarchical circuit comparison. This statement may appear only once in a rule file.

The nmLVS hierarchical SPICE netlister places \*.SEEDPROM statements in all subcircuits that were subject to seed promotion. This is done regardless of the LVS Expand Seed Promotions setting.

This statement operates on hcells specified explicitly (for example, in an hcell list) as well as implicitly (for example, with -automatch). However, **LVS Box** cells are not expanded by this statement.

Hcells expanded by this statement are indicated with warnings in the circuit comparison transcript. For example:

```
WARNING: Cells "nand" and "nand" expanded due to seed promotion -  
correspondence ignored.
```

See also [LVS Show Seed Promotions](#).

#### Examples

```
// Do not expand hcells in which seed promotions occur.  
LVS EXPAND SEED PROMOTIONS NO
```

## LVS Expand Unbalanced Cells

Specification statement

### **LVS EXPAND UNBALANCED CELLS {YES | NO}**

Used only in Calibre nmLVS-H.

#### Parameters

- **YES**

Keyword that instructs Calibre to selectively expand hcells that are instantiated a different number of times in the source and layout. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs Calibre not to selectively expand hcells that are instantiated a different number of times in the source and layout.

#### Description

Specifies whether Calibre nmLVS-H expands hcells that are instantiated a different number of times in the source and layout. This statement is ignored in flat LVS. You may specify this statement once. The default is YES. (See the [Hcell](#) statement for information about defining hcells.)

The following occurs when you specify LVS Expand Unbalanced Cells YES:

When comparing a layout block to a source block, nmLVS-H counts the instances of every hcell placed directly in the block. Hcells that have different numbers of instances in the layout and in the source are usually expanded. Expansion occurs only within that particular block, so that those same hcells may be preserved within other blocks where they are well-behaved. In some cases, hcells are not expanded even if they have different instance counts. Specifically, expansion does not occur if nmLVS-H determines that expanding the cell cannot eliminate the mismatch.

Expansion occurs down to the next level of hcells (not to the bottom); after a cell is expanded, the process is repeated. Cells expanded by this statement are indicated in the session transcript with lines beginning with the words EXPANDING unbalanced cells.

Hcells are hierarchically corresponding cells as specified explicitly with the -hcell command line option or implicitly with the -automatch option. Note that for most LVS runs, -automatch is discouraged.

### Examples

Consider this hierarchy where CHIP is the top-level cell. A, B, and C are cell instantiations with multiple instantiations indicated by “\* N.” Assume that cell C contains five MP devices.

Layout:

CHIP

->A

-->C \* 1

-->MP \* 5

->B

-->C \* 3

Source:

CHIP

->A

-->C \* 2

->B

-->C \* 3

Hcell list:

A A

B B

C C

In cell A, one instance of cell C was implemented directly at the transistor level in the layout. When processing cell A, nmLVS-H expands all instances of cell C. As a result, layout cell A can be verified correctly against source cell A. However, when processing cell B, nmLVS-H uses C as an hcell and does not expand it.

# LVS Filter

Specification statement

```
LVS FILTER component_type [(component_subtype)  
[property_name [(spice_parameter)] [filter_constraint]]  
{SHORT [pin_name ...] | OPEN}  
[SOURCE] [LAYOUT]  
[DIRECT] [MASK]
```

## Summary

Filters out device instances during the LVS comparison phase based upon component type in both source and layout, and leaves the circuit shorted or open, depending on what you specify. Additional parameters make the filter more restrictive.

## Parameters

- *component\_type*

A required component type to which the filter applies. The tool filters out all instances of this component type that meet the other criteria specified in the statement. The *component\_type* may be any SPICE device type, primitive subcircuit name, or any subcircuit name that is used in an **LVS Box** statement. Wildcards are not permitted.

- (*component\_subtype*)

An optional component subtype, which must be in parentheses, to which the filter applies. The filter disregards subtype when you do not specify this parameter.

- *property\_name*

An optional name of a property to which the filter applies. When present, the filter applies only to instances that own the specified property. Otherwise, the filter applies to instances regardless of their properties.

- (*spice\_parameter*)

An optional string in parentheses that is an upper or lowercase letter specified with the *property\_name* parameter. The letter instructs LVS to treat property values as strings in SPICE-like syntax (see section “SPICE-Like Property Syntax” in the *ICverify User’s and Reference Manual*) and to parse the strings to obtain the specified SPICE value. Possible values are as follows:

**W** MOS width

**C** capacitance

**L** MOS length

**A** diode area

**R** resistance

**P** diode perimeter

When this parameter is present, the filter applies only to instances that own a property with the specified *property\_name* parameter. The property value contains the specified *spice\_parameter*.

- *filter\_constraint*

An optional constraint (see the “Constraint Notation” column of [Table 2-2](#) on page 45), specified with the *property\_name* parameter. The constraint limits the value of the property. Either numeric or string constants, numeric variables, or, in Pyxis Layout, process variables can be used in the constraint. A numeric value can be positive or non-positive. String constants are case-insensitive. Interval constraint endpoints *a* and *b* (such as  $> a \leq b$ ), must be both numbers or both strings, and *a* must be less than *b*.

- **SHORT** [*pin\_name* ...] | **OPEN**

A required keyword set that instructs LVS how to treat nets connected to the pins of the instance to which the filter applies. Possible choices are as follows:

**SHORT** — Specifies that nets are shorted together. If specified without a pin list, all nets are shorted.

*pin\_name* — The name of a device pin. A list of pins may be specified. Pin names must exist in the device definition in order for filtering to occur. Standard pin abbreviations for built-in devices are permitted (like p for pos, n for neg, g for gate, and so on). When specified, only the listed pins are shorted together; other pins are left unconnected.

**OPEN** — All nets are left unconnected.

- **SOURCE**

An optional keyword that specifies the statement applies to the source design. This option is used by default.

- **LAYOUT**

An optional keyword that specifies the statement applies to the layout design. This option is used by default.

- **DIRECT**

An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.

- **MASK**

An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

## Description

Filters out source and layout instances that conform to the criteria specified by the statement’s parameters. This statement is typically used to filter SPICE device elements. The filtering applies only during LVS comparison, not netlist extraction.

Either **OPEN** or **SHORT** must be specified. If you specify **OPEN**, then all nets that connect to filtered components are left open. If you specify **SHORT** with no pin list, then all nets that connect to filtered components are shorted together.

If **SHORT** is followed by a *pin\_name* list, LVS shorts together all listed pins. Nets connected to the remaining pins of a filtered instance are left unconnected. Pin names that are also keywords

for the LVS Filter statement must be enclosed in quotes, for example: SHORT IN1 IN2 “SOURCE”.

To filter devices by *property\_name*, the *property\_name* must appear in a [Trace Property](#) statement.

The following consistency checks are implemented for pin names when the rule file is compiled:

1. No pin name may appear more than once in each LVS Filter statement (if case-sensitivity is enabled, names that are different only by case are considered different names and are allowed).
2. If two LVS Filter statements have the same *component\_type* and *component\_subtype*, apply to the same design (LAYOUT or SOURCE), and both specify **SHORT**, then both statements must have identical *pin\_name* lists.
3. If several device pins are swappable, either all must appear in the **SHORT** pin list, or none at all. If the rule file contains DEVICE statements for the device referred to by an LVS Filter statement, this check is performed when the rule file is compiled.

When LVS matches LVS Filter statements to device or cell templates, and the LVS Filter statement has a non-empty list of SHORT pins, LVS applies the filter to a template only if all pins in the list exist on the template (standard abbreviations for built-in device pins, such as p for pos, g for gate, and so forth, are allowed). Pins in the SHORT list that do not exist for the specified device are ignored.

Each Mask-mode LVS Filter specification statement can have a set of corresponding rule file [Device](#) operations; however, Device statements are not required in netlist-to-netlist comparison, for example. Be aware that [Layout Case](#), [Source Case](#), and [LVS Compare Case](#) statements can have an effect on device filtering.

In Calibre, if [Layout System](#) specifies a geometric layout, then a syntactical check is performed on all LVS Filter statements in the LAYOUT set that have a *property\_name* parameter. When the syntactical check is performed, it enforces the following:

- A *spice\_parameter* cannot be used as part of the property being checked because properties for extracted devices do not have SPICE-like values as described previously in the Parameters section.
- There must be at least one Device operation having a component type corresponding to the LVS Filter specification statement in which a *property\_name* parameter appears.
- The *property\_name* parameter must be computed as follows:
  - It must be computed in *all* Device operations that correspond to the LVS Filter specification statement.
  - It must have the correct type (numeric or string) in *all* Device operations that correspond to the LVS Filter specification statement. No string-valued filter constraints are allowed in this situation because extracted devices do not currently compute string-valued properties.

- The *property\_name* must not appear as a vector type in any Device operation that corresponds to the LVS Filter statement. SUM() is an example of a vector function in a Device property calculation.

This check in Pyxis Layout is performed on all LVS Filter statements in the MASK LAYOUT set that have a *property\_name* parameter.

### Examples

#### Example 1

The following statement filters out all mp-type layout instances that own a w property with a value greater than 3 but less than 5. The connecting nets are left open.

```
LVS FILTER mp w > 3 < 5 OPEN LAYOUT
```

#### Example 2

The following statement filters out all c-type source and layout instances. The connecting nets are left open.

```
LVS FILTER c OPEN
```

#### Example 3

The following examples filter resistors with a resistance value equal to zero out of a SPICE netlist:

- For resistor r0 in the SPICE netlist having a zero value:

```
.subckt zcell porta portb
* devices:
r0 porta 4 0 $.MODEL=red
r1 1 portb 0.3333 $.MODEL=blue
r2 1 4 0.3333 $.MODEL=blue
.ends zcell
```

To filter devices by property, the property must be traced:

```
TRACE PROPERTY R(red) r r 0
LVS FILTER R R == 0 SHORT SOURCE
// null value does not equal zero
```

- For all resistors in the SPICE netlist having a model property of red:

```
.subckt zcell porta portb
* devices:
r0 porta 4 0 $.MODEL=red
r1 1 portb 0.3333 $.MODEL=blue
r2 1 4 0.3333 $.MODEL=blue
.ends zcell
```

the corresponding rule file statement is:

```
LVS FILTER R(red) SHORT SOURCE
```

**Example 4**

This example shows how to filter a cell from a source netlist and short the connections to the cell.

```
LVS BOX SOURCE "cellA" // make cellA a box cell in the source  
LVS FILTER "cellA" SHORT SOURCE // filter cellA and short it in the source
```

**Example 5**

The following statement filters out all r type and x subtype source instances that own an instpar property with a resistance value equal to zero. The resistance is specified in the instpar property using SPICE-like syntax. The connecting nets are shorted together.

```
LVS FILTER r(x) instpar(r) == 0 SHORT SOURCE
```

## LVS Filter Unused Bipolar

Specification statement

### LVS FILTER UNUSED BIPOLAR {NO | YES}

#### Parameters

- **NO**

Keyword that instructs the tool not to filter unused bipolar transistors. This is the default behavior if you do not include this statement in your rule file. A NO specification can be overridden by the LVS Filter Unused Option YB statement.

- **YES**

Keyword that instructs the tool to filter unused bipolar transistors.

#### Description

Specifies whether to filter unused bipolar transistors. The default is NO. Can be specified once in your rule file. The filtering applies only during LVS comparison, not netlist extraction.

This statement is equivalent to the statement [LVS Filter Unused Option YB](#), which filters bipolar transistors with base and emitter tied together.

This statement can affect ERC path checking results if [ERC Path Also BIPOLAR](#) is specified.

In Pyxis Layout, this statement sets the value of application variable lvs\_filter\_unused\_bipolar\_transistors.

#### Examples

```
// Do not filter unused bipolar transistors
LVS FILTER UNUSED BIPOLAR NO
```

# LVS Filter Unused Capacitors

Specification statement

## LVS FILTER UNUSED CAPACITORS {NO | YES}

### Parameters

- **NO**

Keyword that instructs the tool not to filter unused capacitors. This is the default behavior if you do not include this statement in your rule file. A NO specification can be overridden by the LVS Filter Unused Option RD RE statement.

- **YES**

Keyword that instructs the tool to filter unused capacitors.

### Description

Specifies whether to filter unused capacitors. The default is NO. Can be specified once in your rule file. The filtering applies only during LVS comparison, not netlist extraction.

This statement is equivalent to the statement: [LVS Filter Unused Option RD RE](#), which filters capacitors with a POS or NEG pin floating and capacitors with POS and NEG pins tied together.

This statement can affect ERC path checking results if [ERC Path Also CAPACITOR](#) is specified.

### Examples

```
// Do not filter unused capacitors.  
LVS FILTER UNUSED CAPACITORS NO
```

## LVS Filter Unused Diodes

Specification statement

### LVS FILTER UNUSED DIODES {NO | YES}

#### Parameters

- **NO**

Keyword that instructs the tool not to filter unused diodes. This is the default behavior if you do not include this statement in your rule file. A NO specification can be overridden by the LVS Filter Unused Option RF RG statement.

- **YES**

Keyword that instructs the tool to filter unused diodes.

#### Description

Specifies whether to filter unused diodes. The default is NO. Can be specified once in your rule file. The filtering applies only during LVS comparison, not netlist extraction.

This statement is equivalent to the statement: [LVS Filter Unused Option RF RG](#), which filters diodes with a POS or NEG pin floating and diodes with POS and NEG pins tied together.

This statement can affect ERC path checking results if [ERC Path Also DIODE](#) is specified.

#### Examples

```
// Do not filter unused diodes.  
LVS FILTER UNUSED DIODES NO
```

# LVS Filter Unused MOS

Specification statement

## **LVS FILTER UNUSED MOS {NO | YES}**

### Parameters

- **NO**

Keyword that instructs the tool not to filter unused MOS transistors. This is the default behavior if you do not include this statement in your rule file. A NO specification can be overridden by the LVS Filter Unused Option AB AC AD F G J L statement.

- **YES**

Keyword that instructs the tool to filter unused MOS transistors.

### Description

Specifies whether to filter unused MOS transistors. The default is NO. Can be specified once in your rule file. The filtering applies only during LVS comparison, not netlist extraction.

This statement is equivalent to the statement: [LVS Filter Unused Option AB AC AD F G J L](#), which filters MOS devices with the following characteristics:

- Source, drain, and gate pins tied together.
- Floating gate pin, and source and drain pins connected to a single power net.
- Floating gate pin, and source and drain pins connected to a single ground net.
- MN and LDDN devices with the gate tied to ground.
- MP and LDDP devices with the gate tied to power.
- Gate tied to either power or ground and the source and drain tied together.
- Either the source or drain is floating.

This statement affects ERC path checking results by filtering out unused MOS devices, which are included in the ERC definition of path.

In Pyxis Layout, this statement sets the value of application variable lvs\_filter\_unused\_mos\_transistors.

### Examples

```
// Do not filter unused MOS devices.
LVS FILTER UNUSED MOS NO
```

# LVS Filter Unused Option

Specification statement

**LVS FILTER UNUSED OPTION** *option* [*option* ...]  
[SOURCE] [LAYOUT]

## Summary

Provides control of unused device filtering. The filtering applies only during LVS comparison, not netlist extraction.

## Parameters

- *option*

A required, case-insensitive keyword that specifies various rules to follow for the filtering of unused devices. You may specify **option** any number of times in one statement, separated by spaces. The list of keywords is as follows:

- AB Filters MOS devices with source, drain, and gate pins tied together.
- AC Filters MOS devices with floating gate pin, and source and drain pins connected to a single power net.
- AD Filters MOS devices with floating gate pin, and source and drain pins connected to a single ground net.
- AE Filters MOS devices with source, drain, and gate pins all floating.
- AF Filters MOS devices with source and drain pins tied together.
- AG Filters MOS devices with all pins tied together, including bulk and optional pins.
- B Filters MOS devices if the gate is floating or has no path to any pad, and either the source or drain is floating.
- C Filters MOS devices with gate tied to power or ground and either the source or drain floating.
- D Filters MOS devices if the gate is floating, either source or drain have a path to power, and neither source nor drain have paths to non-power pads.
- E Filters MOS devices if the gate is floating, either source or drain have a path to ground, and neither source nor drain have paths to non-ground pads.
- F Filters MN and LDDN devices with the gate tied to ground.
- FY Filters MN and LDDN devices with gates tied to ground. Connects source and drain nets if not connected to different pads. If the source and drain nets are connected to different pads then they are left separate and a warning is issued in the LVS report. Because the FY filter creates connections that do not physically occur, caution should be exercised when using this option.
- G Filters MP and LDDP devices with the gate tied to power.

- GY Filters MP and LDDP devices with the gate tied to power. Connects source and drain nets if not connected to different pads. If the source and drain nets are connected to different pads then they are left separate and a warning is issued in the LVS report. Because the GY filter creates connections that do not physically occur, caution should be exercised when using this option.
- H Filters MOS devices with both source and drain tied to power, and series MOS devices with both ends of the series tied to power.
- I Filters MOS devices with both source and drain tied to ground, and series MOS devices with both ends of the series tied to ground.
- IH Filters MOS devices with one of the source/drain pins tied to ground and the other tied to power, and series MOS devices with one end of the series tied to ground and the other end tied to power.
- INV Filters MOS devices which form an inverter with a floating output net. In this case, an inverter is defined as two MOS devices, one of P-type and one of N-type, with shorted gate pins and one source or drain pin of each MOS device connected to the output net of the inverter. Refer to the [Description](#) section for more information on using the INV option. This option is ignored by the EXCLUDE UNUSED option in ERC operations such as [Pathchk](#) and [Device Layer](#).
- J Filters MOS devices with the gate tied to either power or ground and the source and drain tied together.
- K Filters MOS devices with neither source nor drain having any path to any pad.
- L Filters MOS devices if either the source or drain is floating.
- M Filters MP and LDDP devices with the gate tied to ground. Connects source and drain nets if not connected to different pads. If the source and drain nets are connected to different pads, then they are left separate and a warning is issued in the LVS report.
- N Filters MN and LDDN devices with the gate tied to power. Connects source and drain nets if not connected to different pads. If the source and drain nets are connected to different pads, then they are left separate and a warning is issued in the LVS report.
- O Repeats all unused device filtering until no more devices can be filtered. Also repeats series and parallel reduction of capacitor, resistor, diode, and MOS devices, split gate reduction, and semi-series MOS reduction.
- P Filters MOS, bipolar, resistor, capacitor, and diode devices with no *general paths* to any pad (refer to [LVS Filter Unused Option Definitions](#) section). This option filters islands of devices that are isolated from all pads. This option is ignored by the EXCLUDE UNUSED option in ERC operations such as [Pathchk](#) and [Device Layer](#).

- Q Filters MOS, bipolar, resistor, capacitor, and diode devices with no *general paths* to any non-power/ground pads (refer to [LVS Filter Unused Option Definitions](#) section). This option filters all islands of devices that are isolated from all pads, except possibly power/ground pads. Note that, by definition, all devices filtered by option P are also filtered by option Q. This option is ignored by the EXCLUDE UNUSED option in ERC operations such as [Pathchk](#) and [Device Layer](#).
- RB Filters resistors with POS or NEG pin floating.
- RC Filters resistors with POS and NEG pins tied together.
- RD Filters capacitors with POS or NEG pin floating.
- RE Filters capacitors with POS and NEG pins tied together.
- RF Filters diodes with POS or NEG pin floating.
- RG Filters diodes with POS and NEG pins tied together.
- S Filters bipolar transistors with at least two floating pins (collector, base, or emitter).
- T Filters bipolar transistors with emitter and collector tied together and not connected to other devices or pads.
- U Filters MOS devices with both the source and drain floating.
- V Filters MOS devices if the gate is floating, the source or drain is floating, and the non-floating pin is connected to power or ground.
- YB Filters bipolar transistors with base and emitter tied together.
- YC Filters bipolar transistors with collector, base, and emitter tied together.
- ZB Filters resistors with both POS and NEG pins floating.
- ZC Filters capacitors with both POS and NEG pins floating.
- ZD Filters diodes with both POS and NEG pins floating.
- ZE Filters bipolar transistors with collector, base, and emitter pins all floating.

- **SOURCE**

An optional keyword that specifies the statement applies to the source design. This option is used by default.

- **LAYOUT**

An optional keyword that specifies the statement applies to the layout design. This option is used by default.

### Description

Controls the process of filtering out unused devices during LVS comparison. This statement can be specified any number of times and options can be repeated. Options are accumulated from all statements. There is no default.

In nmLVS-H comparison, the design is flattened up to the level of hcells. Unused device filter options are processed locally within hcells.

Unused device filtering occurs in ERC path checking (enabled through [ERC Pathchk](#) or [Pathchk](#)) by ignoring unused devices that are in the definition of a path. (See the following section for the precise definition of a path.)

## Hierarchical Operation

In hierarchical operation, the options B, D, E, H, I, J, K, and T control unused device filtering where connections can be made to certain entities outside of a cell. These external connections could disable a filter if a given device connects to such an entity. When filtering unused devices at the cell level, external connections to such entities cannot be ruled out, so the filter is disabled if such a connection is possible.

Hierarchical ERC path checking treats every cell in the design as an hcell, regardless of whether hcells have been declared. Therefore, hierarchical ERC path checking performs unused device filtering locally within *all cells*. Because hierarchical ERC treats every cell as an hcell, but hierarchical LVS comparison does not, unused device filtering can differ between these applications.

You can specify device filter options from the LVS Options pane in Calibre Interactive - nmLVS. See “[Setting Filter Unused Device Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## LVS Filter Unused Option Definitions

This section defines terminology used in the descriptions of the options.

- MOS devices are component types MP, MN, ME, MD, M, LDDP, LDDN, LDDE, LDDD, LDD, and equivalent device types. Bipolar devices are component type Q and equivalent device types. Resistors are component type R and equivalent device types. Capacitors are component type C and equivalent device types. Diodes are component type D and equivalent device types. Equivalent types are specified in an [LVS Device Type](#) statement. Filtering is only applied to proper built-in devices for comparison and obey the applicable rules regarding the number of pins, pin names, and so forth. See the [Device](#) statement.
- A *path* is any path that leads through source/drain pins of MOS devices or pos/neg pins of resistor devices. Additionally, bipolar, capacitor, and diode devices can be added to the definition of path by using the [ERC Path Also](#) statement. Power and ground nets break paths.
- A *general path* is a path through any pin of any device. Power and ground nets break a general path.
- A *pad* is any power or ground net; any top-level net with a user-given name that appears in both layout and source; any top-level port or (in hierarchical LVS applications); any hcell port.

## LVS Filter Unused Option

---

- A *power pad* within an hcell is any net connected to a top level power net in all placements of the hcell.
- A *ground pad* within an hcell is any net connected to a top level ground net in all placements of the hcell.
- A *floating device pin* is a pin where the net connected to it is not connected to any other instance pins, is not connected to a port of the containing cell, does not serve as an initial correspondence point, and is not a power or ground net. (Cell ports that are ignored or removed for any reason do not count.)

## INV Option Details

The INV option operates independently of gate recognition; that is, inverters with floating output are filtered regardless of whether gate recognition is enabled by the [LVS Recognize Gates](#) statement. Additionally, use of the INV option alone does not filter out inverters whose output net is not floating but is connected to other unused devices. However, enabling both the INV and O options does produce this effect, in addition to causing inverter chains with the floating output of the last inverter to be filtered out.

When using the INV option, the output net must meet all of the following conditions:

- Must not be connected to any other devices
- Must not be an external pin of a cell
- Cannot be a potential initial correspondence point

If all conditions are met, the INV option causes both the P-type and N-type MOS devices to be filtered as unused. Note that the other source and drain pins of the MOS devices do not have to connect to voltage nets.

## Examples

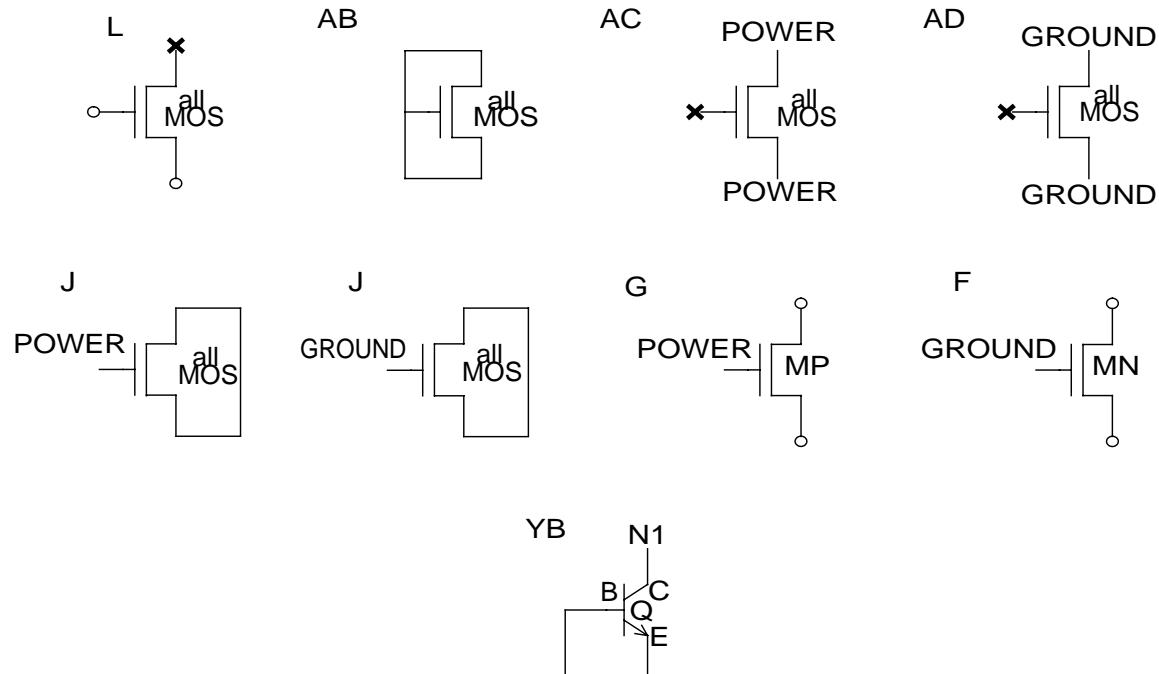
### Example 1

```
LVS FILTER UNUSED OPTION AB RC RE ZD SOURCE  
/* in the source netlist, filter MOS devices with source, gate, and drain  
pins tied together; filter resistors with POS and NEG pins tied together;  
filter capacitors with POS or NEG pins floating; filter diodes with POS  
and NEG pins floating. */
```

**Example 2**

Figure 4-159 shows some common unused device filter options. The “x” denotes a floating pin and the “o” denotes a pin connected to other instances and/or to ports.

**Figure 4-159. Unused Device Filter Option Examples**



## LVS Filter Unused Resistors

Specification statement

### LVS FILTER UNUSED RESISTORS {NO | YES}

#### Parameters

- **NO**

Keyword that instructs the tool not to filter unused resistors. This is the default behavior if you do not include this statement in your rule file. A NO specification can be overridden by the LVS Filter Unused Option RB RC statement.

- **YES**

Keyword that instructs the tool to filter unused resistors.

#### Description

Specifies whether to filter unused resistors. The default is NO. Can be specified once in your rule file. The filtering applies only during LVS comparison, not netlist extraction.

This statement is equivalent to the statement: [LVS Filter Unused Option RB RC](#), which filters resistors with a POS or NEG pin floating and resistors with POS and NEG pins tied together.

This statement affects ERC path checking results by filtering out unused resistor devices, which are included in the ERC definition of path.

#### Examples

```
// Do not filter unused resistors.  
LVS FILTER UNUSED RESISTORS NO
```

## LVS Flatten Inside Cell

Specification statement

**LVS FLATTEN INSIDE CELL {NO | {[SOURCE] [LAYOUT]} {YES | *cell* [*cell* ...]}}}**

Used only in Calibre nmLVS-H and PERC.

### Parameters

- **NO**  
Keyword that instructs the tool not to flatten the hierarchies of hcells in a netlist. This option is used by default.
- **YES**  
Keyword that instructs the tool to flatten the internal hierarchies of hcells in a netlist based upon correspondence with those specified in the [Flatten Inside Cell](#) specification statement. This keyword may not be specified with *cell*.
- ***cell***  
Parameter that instructs the tool to flatten the hierarchy of the specified *cell* in a netlist. More than one *cell* may be specified. To prevent ambiguity, the *cell* parameter must be the final argument in the statement. The asterisk (\*) wildcard is allowed and matches zero or more characters. When using this wildcard, enclose the cell name in quotes. This parameter may not be specified with **YES**.
- **SOURCE**  
An optional keyword that specifies the statement applies to the source design. This option is used by default when **YES** or *cell* is specified. It should not be specified with **NO**.
- **LAYOUT**  
An optional keyword that specifies the statement applies to the layout design. This option is used by default when **YES** or *cell* is specified. It should not be specified with **NO**.

### Description

Specifies whether to flatten the internal hierarchies of hcells down to the device level in the specified netlists during LVS comparison. (This is referred to as cell expansion. Hcell contents affected by this statement are not flattened to the top level, just to the hcell level.)

This statement affects Calibre nmLVS-H netlist comparison and hierarchical PERC applications. Cells that do not have their hierarchies flattened by this statement, and which are not hcells, are expanded. This is the usual behavior in netlist comparison and in PERC. This specification statement has no effect upon layout netlist extraction.

By default, the **NO** option is used and netlist hcell flattening is not performed. The **YES** and *cell* options are useful in cases where device properties such as resistance are calculated between two points, and the only way to calculate the properties is to expand the hierarchies of hcells with the devices.

This statement may appear multiple times in the rule file. A statement that uses the **NO** keyword should not be used together with one that uses **YES** or *cell*.

The SOURCE and LAYOUT keywords specify which netlists have hcell expansion applied to them. By default, both source and layout designs are affected when **YES** or *cell* is specified. (PERC operates only on the source or layout, as specified by the [PERC Netlist](#) specification statement.) Recall that hcell names can differ between layout and source designs. If hcell names differ, care should be taken to ensure the correct cells are expanded.

When **YES** is specified, hcells appearing in the Flatten Inside Cell specification statement are read. The entire contents of the cells in that list are then expanded in the specified netlists.

When the *cell* parameter is used, then the entire contents of the specified cell are expanded. More than one *cell* may be specified.

The aggregate of all LVS Flatten Inside Cell statements in the rule file is used to generate the list of cells that have their internal hierarchies expanded. For example, if you specify this:

```
FLATTEN INSIDE CELL A B
LVS FLATTEN INSIDE CELL YES
LVS FLATTEN INSIDE CELL C
```

then the netlist hierarchies of cells A, B, and C are expanded in both layout and source netlists.

Note that Flatten Inside Cell typically specifies cells that exist in the layout but not necessarily in the source. So in the preceding example, if cells A and B do not exist in the source, then the corresponding names of source hcells are determined from the hcell list and the corresponding cells are expanded in the source. If cell C exists only in the layout or source, then the corresponding hcell name is determined from the hcell list and the corresponding cells are expanded in both designs.

Flattening inside cells is performed at the same time as expansion of unbalanced hcells. If cell flattening is enabled, this transcript message:

```
EXPANDING unbalanced cells ...
```

changes to this:

```
EXPANDING unbalanced and FLATTEN INSIDE cells ...
```

The cells that are expanded are listed in the transcript similarly to the unbalanced cells, for example:

```
EXPANDING unbalanced and FLATTEN INSIDE cells ...
EXPANDING cells inside LAYOUT cell b, SOURCE cell b
    a in b ( 2 instances, LAYOUT )
    a in b ( 2 instances, SOURCE )
Unbalanced and FLATTEN INSIDE cells EXPANDED.
```

By default, matching of cell names is case-insensitive. This can be controlled by the [LVS Compare Case](#) statement.

## Examples

Consider a design that has the following hcells and assume the context is LVS circuit comparison.

LAYOUT SOURCE

B	B1
C	C1

If the rule file contains the following statements:

```
FLATTEN INSIDE CELL B C1 // C1 is a source cell
LVS FLATTEN INSIDE CELL YES
```

then the contents of the hcells B, B1, C, and C1 are expanded during comparison. (Ordinarily, C1 would not be specified in a Flatten Inside Cell statement because it is not a layout cell.) However, if the statements are these:

```
FLATTEN INSIDE CELL B C1 // C1 is a source cell
LVS FLATTEN INSIDE CELL LAYOUT YES
```

then the contents of the hcells B and B1 are expanded, and the cells C and C1 remain intact because the layout name C does not match the specified name C1.

The previous examples assume that the rule file is not used for circuit extraction. In the case of netlist extraction, the contents of the layout cell B would be expanded, but cell C1 would not be found in the layout, so C1 would not be expanded during extraction.

If the rule file is used for both circuit extraction and circuit comparison, but the cell expansion is only to apply during comparison, then the first example can be rewritten as follows:

```
LVS FLATTEN INSIDE CELL B C1
```

In this case, cell B is not expanded during circuit extraction, but during circuit comparison the contents of the hcells B, B1, C, and C1 are expanded.

Both ways of expansion can be combined. If the rule file contains these statements:

```
FLATTEN INSIDE CELL B
LVS FLATTEN INSIDE CELL YES
LVS FLATTEN INSIDE CELL C1
```

In this case, cell B is expanded during circuit extraction, and cells B1 (B was already expanded during netlist extraction), C, and C1 are expanded during circuit comparison.

## LVS Global Layout Name

Specification statement

**LVS GLOBAL LAYOUT NAME** *group\_name* *net\_name* [*net\_name* ...]

Used only in Calibre nmLVS-H and PERC.

### Parameters

- ***group\_name***  
An arbitrary name for a group of global layout signals.
- ***net\_name***  
A name of a layout net. Multiple ***net\_name*** parameters are allowed.

### Description

This statement reports inconsistent connections that involve *global layout signals*. Global layout signals are layout nets that are expected to be texted consistently throughout the design hierarchy. Power and ground supplies are common examples (VDD, VSS). Net names are not case-sensitive by default.

Each LVS Global Layout Name statement specifies a group of layout nets. For example, the statement:

```
LVS GLOBAL LAYOUT NAME POWERS VDD1 VDD2 VDD
```

specifies that nets VDD1, VDD2, and VDD all belong to one group called POWERS. The statement:

```
LVS GLOBAL LAYOUT NAME GROUNDS VSS
```

specifies that net VSS constitutes a group called GROUNDS.

When LVS Global Layout Name specification statements are present in the rule file, the nmLVS-H SPICE netlister checks for connections at different levels of hierarchy between members of dissimilar groups. The extractor also checks for connections between members of a group at a lower level and texted nets that are not members of any group at an upper level.

More specifically, the circuit extractor detects and reports the following connections:

- A net from one group connected to a placement pin from a different group (one or more levels down the hierarchy). For example, net VDD connected to pin VSS.
- A texted net that is not a member of any group, connected to a placement pin from a defined group (one or more levels down the hierarchy). For example, net A connected to pin VSS.

Violations are reported as warnings of type “Conflicting global names” in the Calibre transcript and in the circuit extraction report. All layout locations are in the coordinate space of the cell in which the warning appears. Placement names are relative to that cell as well. See the examples that follow.

Other types of connections are valid and are not reported. Specifically, all these are valid:

- A net connected to a placement pin from the same group. For example, net VDD1 connected to pin VDD.
- Any net connected to a texted placement pin whose name is not a member of any group.
- Any net connected to an unnamed placement pin.
- An unnamed net connected to any placement pin.

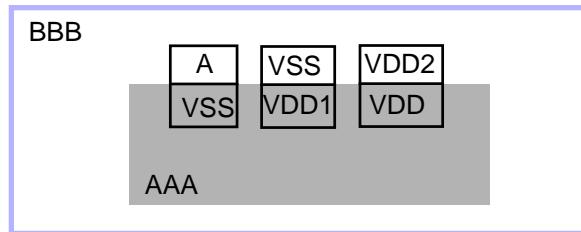
LVS Global Layout Name statements may appear any number of times. Net names are accumulated per group. For example, these three statements:

```
LVS GLOBAL LAYOUT NAME POWERS VDD1
LVS GLOBAL LAYOUT NAME POWERS VDD2
LVS GLOBAL LAYOUT NAME POWERS VDD
```

are equivalent to this:

```
LVS GLOBAL LAYOUT NAME POWERS VDD1 VDD2 VDD
```

Consider the following layout where cell AAA is instantiated in cell BBB:



In this design, it is acceptable to have mixed connections involving VDD1, VDD2, and VDD. The following statements are used:

```
LVS GLOBAL LAYOUT NAME POWERS VDD1 VDD2 VDD
LVS GLOBAL LAYOUT NAME GROUNDS VSS
```

The hierarchical circuit extractor generates the following connectivity warnings:

```
Extraction Errors and Warnings for cell "BBB"
-----
WARNING: Conflicting global names:
Net Id: 1
(1) name "A" at location (0.3,1.6) on layer 1 in this cell
(2) name "VSS" at location (0.3,1.2) on layer 1 in placement
X0 of cell AAA

WARNING: Conflicting global names:
Net Id: 2
(1) name "VSS" at location (0.7,1.6) on layer 1 in this
cell
(2) name "VDD1" at location (0.7,1.2) on layer 1 in
placement X0 of cell AAA
```

## LVS Global Layout Name

---

All locations are in the coordinate space of cell BBB. The placement name, X0, is also relative to cell BBB.

If you want to disallow mixed connections between VDD1, VDD2, and VDD, then you might replace the statements in the previous example with this:

```
LVS GLOBAL LAYOUT NAME P      VDD // keep power names separate
LVS GLOBAL LAYOUT NAME P1     VDD1
LVS GLOBAL LAYOUT NAME P2     VDD2
LVS GLOBAL LAYOUT NAME G     VSS
```

Notes:

- As mentioned previously, this statement operates in the nmLVS-H netlist extractor.
- This statement operates on connectivity that is established in the normal course of circuit extraction in nmLVS-H. This statement, on its own, does not trigger connectivity extraction and does not trigger execution of [Connect](#) operations that would not be executed otherwise.
- When several text objects with the same name appear on an upper-level net, only one representative text object is reported for the net.
- Lower-level cells that are expanded during database construction lose their text objects and so are not checked.
- Conflicts at lower levels of hierarchy can obscure conflicts higher up. For example, if net VDD is connected to pin A of a subcell, and then net A in the subcell is connected to pin VSS lower down, then the A-to-VSS conflict is reported but the VDD-to-VSS conflict is not.

## Examples

```
// Report conflicting global net connections
// VDD, VDD_1, and VDD_2 are PWR nets
LVS GLOBAL LAYOUT NAME PWR VDD VDD_1 VDD_2
// VSS is a GND net
LVS GLOBAL LAYOUT NAME GND VSS
```

## LVS Globals Are Ports

Specification statement

### LVS GLOBALS ARE PORTS {YES | NO}

#### Parameters

- **YES**

Keyword that instructs LVS to treat global nets as ports of the top-level cell. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs LVS not to treat global nets as ports of the top-level cell.

#### Description

Specifies whether LVS treats global nets as ports of the top level cell. The default is YES.

You indicate global nets with .GLOBAL or \*.GLOBAL in SPICE netlists. See “SPICE Format” in the *Calibre Verification User’s Manual*.

The NO option is rarely used, but may be useful in matching layout ports to pins in the top-level .SUBCKT statement in the source, but not necessarily to nets appearing in .GLOBAL statements.

This statement can be specified once in your rule file.

See also [LVS Check Port Names](#), [LVS Ignore Ports](#), and [LVS Spice Override Globals](#).

#### Examples

```
// GLOBAL nets are ports in top-level cells.  
LVS GLOBALS ARE PORTS YES
```

## LVS Ground Name

Specification statement

**LVS GROUND NAME** *name* [*name* ...]

### Parameters

- *name*

A required name of a ground net. You can specify *name* any number of times in one statement. The *name* can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. The *name* can be a string variable.

### Description

Specifies a list of one or more independent ground net names. These names are used by LVS in logic gate recognition, in filtering of unused MOS transistors, and in power supply verification in Direct mode ICtrace. You can specify this statement multiple times in a rule file, and the specified nets are independent of one another.

No ground names are assumed by default, so this statement must be used to declare any ground nets.

The connectivity extractor uses power and ground names to resolve text conflicts (short circuits). Conflicts in which different names are found on a single net are resolved in order of precedence, as follows:

1. Power names, if specified, in rule file order.
2. Ground names, if specified, in rule file order.
3. Alphabetically least name.

In addition, the hierarchical connectivity extractor resolves net name conflicts during high-short resolution in the same manner.

Ground net names are used in [Pathchk](#) and [ERC Pathchk](#) operations for path tracing. They are also used for filtering of unused devices in [Device Layer](#), Pathchk, and ERC Pathchk operations.

The “[Power and Ground Nets](#)” section of the *Calibre Verification User’s Manual* discusses how such nets are used in LVS comparison.

A ground name is considered badly-formed if it is classified as a non-user-given net name by LVS criteria in both layout and source. Such names should not appear in LVS Ground Name statements or related Pyxis Layout variable settings. The section “User-Given Names” in the *Calibre Verification User’s Manual* discusses user-given (as opposed to system-generated) net names.

You can specify power supply management options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Setting Power Supply Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In Pyxis Layout, this statement sets the value of application variable lvs\_ground\_names.

See also [LVS Power Name](#), [LVS Recognize Gates](#), [Pathchk](#), [LVS Cell Supply](#), [LVS Filter Unused MOS](#), and [LVS Filter Unused Option](#).

## Examples

```
// VSS and VSS1 are not virtually connected; these are ground net names  
LVS GROUND NAME VSS VSS1
```

## LVS Heap Directory

Specification statement

**LVS HEAP DIRECTORY “*filename* [*number*]”**

### Parameters

- *filename*

A required directory filename. The specified parameters must be enclosed in quotation marks (“ ”).

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

- *number*

An optional string that specifies the size limit, in Megabytes, for the heap directory.

### Description

---

**Note**

 This statement is supported for backward-compatibility only. In general, you should avoid using this statement in your rule file.

---

Specifies the directory where temporary heap files are to be stored and the directory’s capacity. If *number* is not specified, then directory size is limited only by the space available in the respective disk partition. Multiple statements are allowed and any particular directory may be specified once in your rule file.

---

**Note**

 If the LVS run terminates abnormally, files created in this directory may not be removed and should be manually deleted.

---

This statement is used for large data structures in the LVS comparison and Cnet generation modules, as well as for some data structures in the LVS hierarchical connectivity extractor and short isolation. For these, LVS will use up LVS Heap Directory space before it allocates space in virtual memory. The temporary heap files are regular disk files and thus do not consume swap space. However, those files are memory mapped; often operating system limits exist that control the maximum amount of memory-mapped space allowed per process. On many 32-bit operating systems the limit is close to 2 GB, in addition to another 2 GB allowed for regular virtual-memory heap allocation.

This statement is designed for use with 32-bit operating systems that do not allow a process to allocate a full 4 GB in regular virtual memory heap space but do allow allocation of additional memory using memory mapped files.

This statement has little benefit in operating systems where Calibre can allocate a full 4 GB in regular virtual memory heap space, such as in Solaris 2.6 and higher. This statement also has little benefit on Linux, because process size limits apply globally to all parts of the process.

### Examples

```
// Use this for heap files. Maximum size is 900 MB.  
LVS HEAP DIRECTORY "/scratch2/tmp 900"
```

## LVS Ignore Ports

Specification statement

### LVS IGNORE PORTS {NO | YES}

#### Parameters

- **NO**

Keyword that instructs LVS to consider ports during comparison. This is the default behavior if you do not include this statement in your rule file.

- **YES**

Keyword that instructs the tool to ignore ports.

#### Description

Specifies whether the LVS comparison module should ignore layout and source ports at the top level of the design. This variable controls whether LVS uses such ports as initial correspondence points, and whether it reports any discrepancies involving these ports. Note that the value of this variable *does not* affect connectivity extraction.

This statement does not check port names, just the existence of ports. To compare port names, use [LVS Check Port Names YES](#).

You can specify port behavior from the LVS Options pane in Calibre Interactive—nmLVS. See “[Ignoring Pins](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

This statement can be specified once in your rule file. In Pyxis Layout, this statement sets the value of application variable lvs\_ignore\_ports.

See also [LVS Non User Name](#), [LVS Ignore Trivial Named Ports](#), and [LVS Globals Are Ports](#).

#### Examples

```
// LVS should compare the existence of top-level ports.  
LVS IGNORE PORTS NO
```

# LVS Ignore Trivial Named Ports

Specification statement

## LVS IGNORE TRIVIAL NAMED PORTS {NO | YES}

### Parameters

- **NO**

Keyword that instructs the tool to retain for comparison ports to lower-level cells that do not connect to any devices in a cell (trivial ports). This is the default behavior if you do not include this statement in your rule file.

- **YES**

Keyword that instructs the tool to ignore trivial ports to lower-level cells.

### Description

Controls the handling of ports to lower-level cells that are not connected to any devices, and that have corresponding names in both the source and layout.

This statement does not apply to ports of the top-level cell.

When NO is specified, trivial named ports are not ignored, and become initial correspondence points in the comparison. For example, consider the following design:

```
-- Layout Netlist

.SUBCKT TOP
C0 1 1
X0 1 2 A          $ net 2 is floating
X1 1 3 A          $ net 3 is floating
.ENDS

.SUBCKT A P1 P2
C0 P1 P1          $ P1 is attached to the capacitor
.ENDS

-- Source Netlist

.SUBCKT TOP
C0 1 1
X0 1 2 A          $ net 2 is floating
X1 1 3 A          $ net 3 is floating
.ENDS

.SUBCKT A P2 P1
C0 P2 P2          $ P2 is attached to the capacitor
.ENDS
```

When the netlists are compared, if cell A is an hcell, ports P1 and P2 are identified as initial correspondence points. Since P1 is connected to the capacitor C0 on the layout side for A and P2 is connected to C0 on the source side, a discrepancy will result.

## LVS Ignore Trivial Named Ports

---

When YES is specified, trivial named ports are ignored even if there are similarly named ports in the corresponding source or layout netlist. Assuming YES is specified, in the preceding example the trivial ports (P2 in the layout and P1 in the source) are discarded and the source port P2 is matched with the layout port P1. Thus, the comparison result is CORRECT.

Note that pins on [LVS Box](#) cells are never considered floating. Also, since trivial port removal occurs before filtering of unused devices, ports that are rendered trivial by unused device filtering remain in the design during comparison.

See also [LVS Ignore Ports](#) and [LVS Check Port Names](#).

## Examples

```
// Ignore lower-level ports that do not connect to any devices in a cell
LVS IGNORE TRIVIAL NAMED PORTS YES
```

# LVS Inject Logic

Specification statement

## LVS INJECT LOGIC {YES | NO}

Used only in Calibre nmLVS-H and PERC.

### Parameters

- **YES**

Keyword that instructs nmLVS-H to replace common logic elements with new, primitive elements. This is the behavior if you do not explicitly use this option in your rule file, and the XRC or CCI keyword is not used with [Mask SVDB Directory](#).

- **NO**

Keyword that instructs nmLVS-H not to replace common logic elements.

### Description

Specifies whether nmLVS-H should internally substitute logic in the design. Logic injection is an algorithm in hierarchical circuit comparison that is designed to reduce memory consumption by replacing common logic circuits with new, primitive elements. This process is fully described under “[Logic Injection](#)” in the *Calibre Verification User’s Manual*.

Logic injection is unrelated to hierarchy injection.

The best-case candidates to benefit from logic injection are designs with significant embedded memory content, where the embedded memory blocks have similar sizes, serve as hcells, but are, in-turn, processed mostly at transistor level (that is, they contain no significant hcells inside). Note that if you already have a good [Hcell](#) list, then logic injection may provide no benefit; in some cases, there may even be a slight penalty in run time or memory consumption.

The extracted layout netlist and source netlist are not changed by logic injection. To see the netlists nmLVS-H uses internally when logic injection is enabled, see [LVS Write Injected Layout Netlist](#) and [LVS Write Injected Source Netlist](#).

Logic injection is not compatible with Calibre xRC. If XRC is specified for the Mask SVDB Directory statement, LVS Inject Logic does not replace common logic elements unless YES is explicitly set.

Logic injection is not used by default if you use the Mask SVDB Directory CCI option. This is done to prevent the internal nets of injected circuits from being removed by the injection. If you specify YES explicitly in this case, then logic injection is performed.

LVS Inject Logic YES causes certain logic gate structures not to be recognized when logic gate recognition is enabled. See “[Logic Injection and Gate Recognition](#)” and “[Effects of Logic Injection](#)” in the *Calibre Verification User’s Manual* for more information.

If [LVS Recognize Gates](#) WITHIN TOLERANCE is specified and an injection structure violates the tolerance in any pertinent [LVS Recognize Gates Tolerance](#) statement, then the input pins of the injected circuit are not swappable. Note that circuits with topological swappability, such as

## LVS Inject Logic

---

SRAM bit cells or parallel device groups, are not affected by this tolerance check since their pins are always swappable.

### Examples

```
// Perform logic injection during nmLVS-H.  
LVS INJECT LOGIC YES
```

# LVS Isolate Shorts

Specification statement

## **LVS ISOLATE SHORTS NO**

## **LVS ISOLATE SHORTS YES**

[BY CELL [ALSO]] [BY LAYER [ALSO]]  
 [NO CONTACTS] [UNMERGED] [ACCUMULATE]  
 {[CELL {PRIMARY | ALL} *operand NAME text\_name [text\_name ...]*] | [FLAT]}

### **Summary**

Enables or disables short isolation in the layout, and specifies the format of the short isolation database. Unlike most SVRF statements, the order of the parameters is important in this statement.

### **Parameters**

- **NO**

Keyword that instructs nmLVS not to perform short isolation. This is the default behavior if you do not include this statement in your rule file

- **YES**

Keyword that instructs nmLVS to perform short isolation.

- **BY CELL [ALSO]**

An optional keyword set that instructs nmLVS-H to report the cell of origin for every polygon written to the short isolation database. In most circumstances, this option is preferable over the UNMERGED option. The BY CELL keyword can be used with BY LAYER, in which case the output reflects both modes. See the [BY CELL Keyword](#) section for a complete discussion.

ALSO — An optional keyword that specifies BY CELL output should be included in the results database *in addition to* other representations of the same output data for a given short. See [Table 4-24](#).

- **BY LAYER [ALSO]**

An optional keyword that specifies to generate output in separate results per short, per layer. This option is especially useful in power-ground short debugging. You should use this option unless you have good reasons not to. The BY LAYER keyword can be used with BY CELL, in which case the output reflects both modes. See the [BY LAYER Keyword](#) section for a complete discussion.

ALSO — An optional keyword that specifies BY LAYER output should be included in the results database *in addition to* other representations of the same output data for a given short. See [Table 4-24](#).

- **NO CONTACTS**

An optional keyword that specifies to omit contact layers from the output. Contact layers are layers specified after the BY keyword in [Connect](#) operations.

- **UNMERGED**

An optional keyword that instructs nmLVS-H not to merge segmented polygons prior to output. Normally, hierarchical short isolation segments polygons for processing and then merges them before output. This keyword must appear before the NAME keyword if both are specified. This option is rarely used. This option has no effect in flat short isolation.

- **ACCUMULATE**

An optional keyword that instructs the tool to identify a common path, if it exists, shared by multiple paths between shorted text objects. This option increases run time, as it requires extra search time to find multiple paths sharing a common path. This can be mitigated by using distributed processing and hyperscaling. This keyword may not be specified with FLAT. See the [ACCUMULATE Keyword](#) section for details.

- **CELL {PRIMARY | ALL}**

An optional keyword set that specifies where LVS-H isolates shorts between text objects.

CELL PRIMARY — Isolates shorts between text objects in the primary cell. If the CELL option is not specified, then CELL PRIMARY is used.

CELL ALL — Isolates shorts between text objects in all cells. In each cell, Calibre isolates shorts in that cell, or its sub-hierarchy, that involve text objects local to that cell. This option is best used if texting of net names is consistent for all cells in the design.

This parameter is ignored in flat applications and cannot be specified with the FLAT parameter.

- *operand*

An optional string that indicates logical symbols to combine the CELL and NAME parameters. Possible choices are:

&& — Isolates a short if it satisfies both the CELL *and* NAME parameters: same as Boolean AND.

|| — Isolates a short if it satisfies the CELL *or* NAME parameters: same as Boolean OR.

See [Example 2](#) and [Example 3](#).

- **NAME *text\_name***

An optional keyword, followed by one or more strings, which instructs the tool to isolate shorts between the specified names. The *text\_name* specifies a text object name, which can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Specifying a single *text\_name* parameter only makes sense if you use a wildcard in the *text\_name*, and this matches more than one net name.

When you specify the NAME parameter, then CELL and *operand* parameters must precede the NAME parameter.

If you do not include the NAME parameter, the default behavior of this parameter is:

```
&& NAME "?"
```

and the tool processes all text object names.

Flat Calibre applications ignore the NAME parameter.

- **FLAT**

An optional keyword that instructs the tool to perform the short isolation stage flat, even when executing hierarchical circuit extraction.

When you specify the FLAT parameter, you cannot specify the ANNOTATE or CELL parameters.

## Description

Shorts occur when there are conflicting text objects on the same net. When YES is specified, short isolation finds and outputs the shortest path between two conflicting text points. If there are no shorts during connectivity extraction, then short isolation is not executed. The default setting is NO. The statement can be specified once in your rule file.

When YES is specified, the [Layout System](#) must be geometric. Short isolation is not performed if the Layout System is SPICE or Cnet. Short isolation is executed in flat Calibre nmLVS, layout-to-Cnet translation, and hierarchical circuit extraction in nmLVS-H. Any of these commands are used to perform short isolation in LVS:

```
calibre -lvs rules
calibre -lvs -tl lay.cnet rules
calibre -spice lay.net ... rules
calibre -lvs -hier -hcell cells rules // when Layout System is geometric
```

The first two commands perform flat short isolation. The last two commands perform short isolation hierarchically. Short isolation is also performed in ICtrace.

You can specify short isolation using Calibre Interactive—nmLVS. See “[Invoking Short Isolation](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*. When specifying short isolation in Calibre Interactive, use the **Output shorts by layer** option to specify BY LAYER (recommended).

Shorts are isolated between text objects in the primary cell unless CELL ALL is specified. Processing is independent of [Text Depth](#) in all cells, except the top-level cell, where text objects selected from the Text Depth setting are treated as if they are at the top level.

Short isolation always tries to isolate as many shorts as possible in one run. However, it may not be able to isolate all shorts at once. When using the NAME option, increase the number of *text\_name* parameters to increase the number of shorts isolated.

If more than one path exists between a pair of conflicting text points, then Calibre selects the shortest path. In flat operation, distance is measured by number of polygons in the path. Polygons are merged. This is not necessarily always the geometrically shortest distance. In hierarchical operation, Calibre selects paths that consist of shapes at higher levels of hierarchy over those that pass through cell placements. Such paths may not be the shortest in terms of

polygon count. Thus, different paths may be chosen when operating hierarchically and flat. Within a given level of hierarchy, paths with fewer polygons are selected.

In hierarchical short isolation, the flat short isolation engine may be used for shorts on small nets. This is an internal performance optimization. The hierarchical short isolation is always used for at least the first ten shorts and on all shorts involving large nets.

Also see the sections [Redundant Short Isolation](#) and [Short Isolation and Global Layout Names](#) for more details on the short isolation process.

---

### Note



In nmLVS-H, the LVS Isolate Shorts specification statement does not process virtual connections created by the [Virtual Connect Box Colon](#) and [Virtual Connect Box Name](#) specification statements.

---

Short isolation results can be viewed as soon as they are created, even before the Calibre run terminates. This is indicated in the session transcript as follows:

```
SHORT ISOLATION started.  
SHORT ISOLATION completed. CPU TIME = 0  REAL TIME = 0  ...  
SHORT ISOLATION RESULTS DATABASE = lvs.rep.shortcuts
```

LVS Isolate Shorts finds shorts between texted nets. When trying to isolate a short on an untexted layout net, the [Isolate Path on Layout Net](#) feature in Calibre RVE for LVS may be helpful.

**Pyxis Layout operation** — The short isolation module does not process explicit connections formed by Pyxis Layout multi-shape ports. Shorts caused by multi-shape ports are not isolated.

## Results Output

In Calibre, the short isolation results are written in nmDRC-ASCII database format to the file *lvs\_report\_name.shortcuts* if an [LVS Report](#) name was supplied, or *lvs.layout\_cell\_name.rep.shortcuts* otherwise. The *layout\_cell\_name* is the cell name specified by the [Layout Primary](#) specification statement.

---

### Note



If LVS Isolate Shorts YES is specified and no shorts are found during a run, Calibre nmLVS removes the short isolation result file written during any previous runs. If LVS Isolate Shorts YES is not specified in the rule file, the short isolation result file is *not* removed.

---

The result file can be viewed with any application that can read nmDRC-ASCII files, such as Calibre RVE or Pyxis Layout. See “[Isolating Shorts Interactively](#)” in the *Calibre Interactive and Calibre RVE User’s Manual* for information on how to debug shorts in RVE. You can also virtually repair a short in Calibre RVE for LVS, then run short verification; see “[Verifying Short Repairs](#)”.

All short isolation results are returned in the coordinate space of the top-level cell. In hierarchical operation, results from lower-level cells are shown only once. The lowest, leftmost placement of a cell in the design is used as representative for shorts occurring in that cell.

In the results database, each short appears as a separate nmDRC-style check. The check name has the following format:

```
SHORT <number>. <net>-<net>-...-<net> [in <cell>] [BY CELL] [BY LAYER]
[(<layer>)]
```

where the elements are defined as follows:

<number>	Serial number for the short.
<net>	Names of the nets that are shorted together.
<cell>	Name of the cell where the short occurs. Included in hierarchical operation only.
BY CELL	Included for hierarchical operation when the result is listed according to the BY CELL keyword.
BY LAYER	Included for hierarchical operation when the result is listed according to the BY LAYER keyword.
<layer>	Layer name or number. Included if BY LAYER is specified.

For example, for flat short isolation with BY LAYER:

```
SHORT 1. vss - vcc (metal2)
```

For hierarchical short isolation with BY LAYER:

```
SHORT 1. vss - vcc cell_a BY LAYER (metal2)
```

Each check contains a check comment (called check text) with information about text objects involved in the particular short. For each text object, the text value, location, and layer are indicated. For example:

```
SHORT 1. gnd! - vdd! in BUF4 BY LAYER (Metal2)
2 2 3 May 22 08:29:18 2008
2 Shorted texts:
"gnd!" at (59.27,154.052) on layer "M2_txt" SN 1
"vdd!" at (31.36,149.892) on layer "M2_txt" SN 21
```

For hierarchical short isolation, the sequence number of the polygon in the short is reported with the SN property.

The bulk of data in each check consists of a list of polygons that describe a piecewise linear path connecting the text points in question.

See “ASCII nmDRC Results Database Format” in the *Calibre Verification User’s Manual* for a description of the database format.

**Pyxis Layout operation** — In naming the output database file in Pyxis Layout, *layout\_cell\_name* is the name of the top-level cell associated with the active IC Station window (regardless of the editing context). The result file can be loaded and the shorts viewed using the ICrules options for restoring nmDRC databases and then viewing nmDRC checks.

### NO CONTACTS Keyword

Contact polygons are included in the output unless this is explicitly disabled with the NO CONTACTS option. Specifically, short circuit paths identified by short isolation include a representative contact polygon for each connection point in the short that was formed with a Connect BY operation. When a connection between a particular pair of polygons is formed by more than one contact, only one of those contacts is included in the output and the others are omitted.

Note that short isolation does not isolate shorts that are caused by text labels that are attached to contact polygons. This is true both with and without the NO CONTACTS option.

### BY CELL Keyword

This option enables the reporting of the cell of origin for every polygon written to the short isolation database. The cell of origin is reported using the “CN” property in the database. Short isolation using the BY CELL mode reports the transform to top-level coordinates from the particular placement of the cell that contains the current polygon. For example:

```
CN NAND2_2 c 0 1 -1 0 456 377
```

If BY CELL mode is enabled during short isolation and the UNMERGED option is not specified, reported polygons are merged within each cell, but polygons from different cells are not merged. This is different from the regular output where all polygons in the short path are merged. The main reason for using the UNMERGED report is to avoid merging polygons that originate from different cells. You should use BY CELL mode instead of UNMERGED mode in most cases because BY CELL results in smaller output files that contain more information.

BY CELL mode can be combined with BY LAYER mode. In this case the short isolation report contains a separate check for each layer, and polygons within each check are annotated with “CN” properties.

If you specify BY CELL ALSO, then the short isolation database will contain more than one result for the same short. For example, if you have a vcc-to-vss short in the cell top, then two results are written:

```
SHORT 1. vcc - vss in top
...
SHORT 1. vcc - vss in top BY CELL
```

Similar reports occur if you specify BY LAYER ALSO.

[Table 4-24](#) shows which results are generated depending on the combination of BY CELL and BY LAYER options used (BY CELL checks have CN properties attached to polygons).

**Table 4-24. BY CELL and BY LAYER Output Options**

BY CELL options	BY LAYER options	Results
none	none	fully merged for all layers, no CN properties.
BY CELL	none	BY CELL for all layers.
BY CELL ALSO	none	both fully merged and BY CELL for all layers.
none	BY LAYER	BY LAYER, no CN properties.
none	BY LAYER ALSO	both fully merged and BY LAYER, no CN properties.
BY CELL	BY LAYER	BY CELL BY LAYER.
BY CELL ALSO	BY LAYER	both BY LAYER with no CN properties and BY CELL BY LAYER.
BY CELL	BY LAYER ALSO	both BY CELL for all layers and BY CELL BY LAYER
BY CELL ALSO	BY LAYER ALSO	all of these: fully merged for all layers, BY CELL for all layers, BY LAYER with no CN properties, and BY CELL BY LAYER



**Tip:** When using the BY CELL keyword and viewing results in RVE, it is best to disable the **Highlight in context** option in order to give the best results presentation.

## BY LAYER Keyword

In general, you should always use the BY LAYER keyword in your rule file. BY LAYER ALSO has a similar behavior to BY CELL ALSO in that more than one result presentation for a given short is output to the results database. See [Table 4-24](#) for details about the interactions of these keywords.

When the BY LAYER parameter is specified, output is generated in separate checks per short and per layer. A result contains all shapes on the layer that describe the given short. For example:

```
SHORT 1. VDD - VSS in AVMRAM BY LAYER (MT2)
1 1 3 Feb 2 11:16:59 2001
2 Shorted texts:
"VDD" at (5642.3,520.1) on layer "MT2LG"
"VSS" at (5651.1,519.95) on layer "MT2LG"
```

## LVS Isolate Shorts

---

In this example, the short occurs on layer MT2, between text objects “VDD” and “VSS”. The location and layer of the text objects is given. In this case, the text objects are on a different layer from the layer being reported for this short.

When BY LAYER is not specified, the short result includes all layers.

## ACCUMULATE Keyword

For cases where multiple paths exist between shorted text points, the use of the accumulate mode can further localize the source of the short. In general, if multiple paths exist between shorted text points, the part common to all the paths most likely contains the short. The ACCUMULATE mode identifies this common path if it exists. Note that a short is not guaranteed to be in the common path and may lie elsewhere especially if there are multiple shorts.

The short isolation results database file accounts for the results in ACCUMULATE mode. The SHORT N line includes either a VARIANT N or COMMON label. For example:

```
SHORT 1 (VARIANT 1). VDD - VDDIO in CHIP
SHORT 2 (VARIANT 2). VDD - VDDIO in CHIP
SHORT 3 (COMMON). VDD - VDDIO in CHIP
```

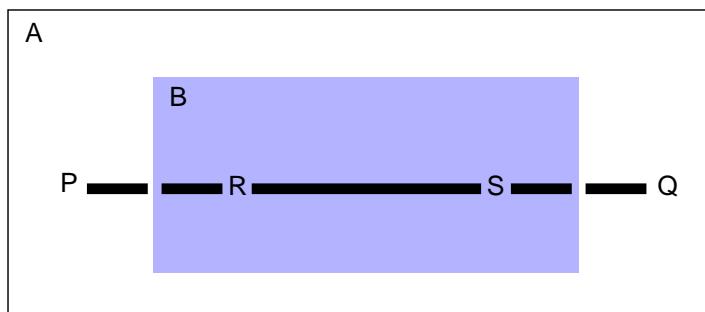
Each VARIANT N path is a distinct path. A maximum of five such paths will be attempted per short. If a common path exists, it is listed separately using the COMMON label.

## Redundant Short Isolation

When the CELL ALL option is specified, hierarchical short isolation does not attempt to isolate a short in a higher-level cell if that short is caused by a short in a lower-level cell that was already isolated. This optimization eliminates redundant reporting of shorts and may significantly reduce hierarchical short isolation run time.

Consider the following example:

**Figure 4-160. Redundant Short Elimination**



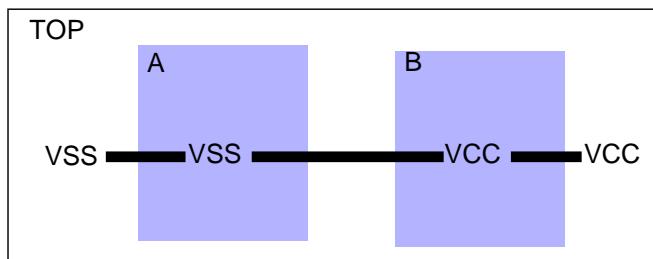
In Figure 4-160, nets R and S are shorted in cell B. This also causes a short between nets P and Q in cell A. Hierarchical short isolation with the CELL ALL option shows the path from R to S in cell B. It does not show the path from P to Q in cell A because that is redundant. However, if

cell A contained an additional independent path also connecting P and Q, then that path would be shown also.

### Short Isolation and Global Layout Names

Short isolation in nmLVS-H considers global signals when isolating shorts. Global signal checking is enabled by the [LVS Global Layout Name specification statement](#). Short isolation uses global signals as additional texts if at least one of the shorted names is a global name. As a result, short isolation may report a shorter path that connects global texts in lower-level cells instead of a longer path that connects top-level shorted texts.

In the following diagram, the names VCC and VSS in the top cell are shorted. Without global names, short isolation finds the path connecting top-level VCC and VSS texts (the outer texts in the diagram).



If VCC and VSS are declared global names as follows:

**LVS GLOBAL LAYOUT NAME POWERS VCC**

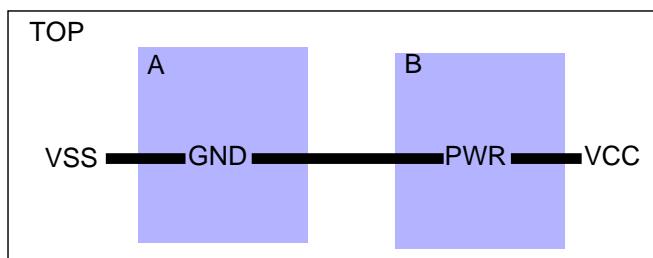
**LVS GLOBAL LAYOUT NAME GROUNDS VSS**

then, in addition to the short-circuit warning, a global names warning is issued. Short isolation internally adds two more texts to the top cell when short isolation finds the short; the texts are VSS from cell A and VCC from cell B, both translated to the correct coordinates in the top cell. With the extra texts, short isolation reports a shorter path between the two inner texts in the diagram.

In the following figure, the names VCC and VSS in the top cell are shorted and there is a global signal conflict. The conflicting global names are not only VCC and VSS but also PWR and GND, due to these rule file statements:

**LVS GLOBAL LAYOUT NAME POWERS VCC PWR**

**LVS GLOBAL LAYOUT NAME GROUNDS VSS GND**



In this case, short isolation again uses the global names, but the names in each group are converted to the *representative name*, which is the name involved in the short. For example, in the POWERS group the representative name is VCC. When the text corresponding to the global name PWR is internally added to the top cell, it is replaced by VCC. Similarly for the GROUNDS group. As a result, the short isolation reports a short between VCC and VSS, but the path actually connects the locations of the global names PWR and GND (the lower level texts in the diagram).

Note that the global signal conflicts are used to add more texts internally to short isolation, not to isolate additional shorts. That is, if there are no short-circuit warnings but only global signal conflict warnings, short isolation is not performed.

For additional discussion of this topic, see “[Short Isolation](#)” in the *Calibre Verification User’s Manual*.

### Examples

All examples assume hierarchical Calibre operation.

#### Example 1

The next statement isolates all shorts in all cells. The results presentation has the maximum flexibility of viewing perspectives on account of the BY LAYER ALSO and BY CELL ALSO keywords. The keyword CELL ALL is best used when text is consistent in all cells.

```
LVS ISOLATE SHORTS YES BY LAYER ALSO BY CELL ALSO CELL ALL
```

#### Example 2

The following statement isolates shorts in the top-level cell between all names OR between the specified names in all cells. Results are organized per layer.

```
LVS ISOLATE SHORTS YES BY LAYER CELL PRIMARY || NAME "VCC?" "VSS?"
```

#### Example 3

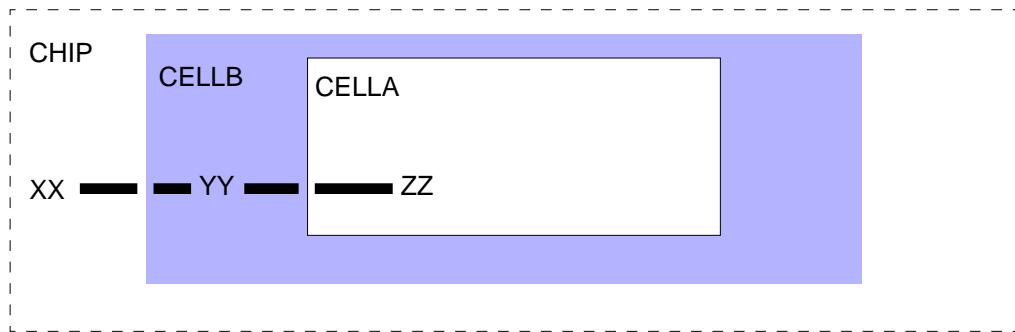
The following statement isolates shorts in all cells only between the specified names. The results presentation has the maximum flexibility of viewing perspectives on account of the BY LAYER ALSO and BY CELL ALSO keywords. The keyword CELL ALL is best used when text is consistent in all cells.

```
LVS ISOLATE SHORTS YES BY LAYER ALSO BY CELL ALSO CELL ALL && NAME "VCC?"  
VSS?"
```

**Example 4**

In Figure 4-161, if the CELL ALL option is used with a Text Depth PRIMARY statement, then no shorts are indicated because no text conflicts exist in any one cell.

**Figure 4-161. No Shorts in Any Cell**



Adjusting the Text Depth allows you to isolate shorts to the appropriate cell level.

# LVS Map Device

Specification statement

**LVS MAP DEVICE** *old\_component\_type* [(*old\_subtype*)]  
  *new\_component\_type* [(*new\_subtype*)]  
  [SOURCE] [LAYOUT]

Used only in Calibre nmLVS and PERC.

## Parameters

- ***old\_component\_type***

Required name of a [Device](#) element type for which a device mapping is applied.

- (*old\_subtype*)

Optional name in parentheses of a component subtype to be mapped to a new name. Omitting this parameter assumes an empty subtype. If this parameter is omitted, then *new\_subtype* must be specified.

- ***new\_component\_type***

Required name of a [Device](#) element type that specifies the component type after the mapping is applied. This parameter may be identical to *old\_component\_type* if only subtypes are being mapped.

- (*new\_subtype*)

Optional name in parentheses of a new device subtype after mapping. Omitting this parameter assumes an empty subtype. If this parameter is omitted, then *old\_subtype* must be specified. This parameter may be identical to *old\_subtype* if only component types are being mapped.

- SOURCE

An optional keyword that specifies the statement applies to the source design. This option is used by default.

- LAYOUT

An optional keyword that specifies the statement applies to the layout design. This option is used by default.

## Description

This statement causes Calibre nmLVS to map devices of one component type and/or subtype to another component type and/or subtype in the specified design (source, layout, or both). This statement applies only to the LVS comparison module, not the device extractor used during connectivity extraction.

Each statement applies to devices whose type and subtype exactly match *old\_component\_type* and *old\_subtype*. An empty subtype matches only an empty subtype for the purpose of device mapping. For each *old\_component\_type* and *old\_subtype* there can be only one LVS Map Device statement referring to each subdesign (layout or source).

The *old\_component\_type* and *old\_subtype* together must be different from the *new\_component\_type* and *new\_subtype*, although the types can be identical or the subtypes can be identical. If the types are different, then only *compatible types* can be mapped from one another. More specifically, you can use this statement to do the following:

- Map a MOS type to a different MOS type.
- Map a user-defined type to a different user type.

Mapping incompatible device types is not allowed; for example, mapping component type C to D is not allowed.

The subtypes remapped by LVS Map Device are not subject to further remapping, that is, device mapping is not applied recursively.

If mapping is applied, then [Trace Property](#) statements that apply to mapped device subtypes must reference the *new names*. Similarly any statements from the LVS Reduce family that apply to mapped device subtypes must reference the *new\_subtype* name.

## Examples

### Example 1

Layout devices MP(PA), MP(PB), and MP(PC) are all mapped to MP(P). The source is expected to contain devices MP(P). This is an example of N-to-1 mapping.

```
LVS MAP DEVICE MP(PA) MP(P) LAYOUT
LVS MAP DEVICE MP(PB) MP(P) LAYOUT
LVS MAP DEVICE MP(PC) MP(P) LAYOUT
```

### Example 2

Devices C with empty subtype are mapped to C(A) in both layout and source. Devices C(XYZ) with non-empty subtype XYZ are unaffected.

```
LVS MAP DEVICE C C(A)
```

### Example 3

Devices C(A) are mapped to C with empty subtype in both layout and source. Devices C with empty subtype are unaffected.

```
LVS MAP DEVICE C(A) C
```

### Example 4

Devices MP(A) are mapped to MP(B) in the layout and to MP(C) in the source.

```
LVS MAP DEVICE MP(A) MP(B) LAYOUT
LVS MAP DEVICE MP(A) MP(C) SOURCE
```

### Example 5

Type ME devices are mapped but subtypes remain unchanged:

```
LVS MAP DEVICE ME(e_nfet) MN(e_nfet)
LVS MAP DEVICE ME(e_pfet) MP(e_pfet)
```

### Example 6

Type ME devices are mapped and subtypes also:

```
LVS MAP DEVICE ME(e_nfet) MN(n)
LVS MAP DEVICE ME(e_pfet) MP(p)
```

### Example 7

This combination of LVS Map Device statements is not allowed because conflicting mappings are requested for device MP(A) in the layout.

```
LVS MAP DEVICE MP(A) MP(B) LAYOUT
LVS MAP DEVICE MP(A) MP(C) LAYOUT // error. Conflicting mapping in layout.
```

### Example 8

Devices MP(A) in layout are mapped to MP(B). They are not further mapped to MP(C). However, if the layout, before mapping, contains devices MP(B), these devices are mapped to MP(C).

```
LVS MAP DEVICE MP(A) MP(B) LAYOUT
LVS MAP DEVICE MP(B) MP(C) LAYOUT
```

# LVS MOS Swappable Properties

Specification statement

## LVS MOS SWAPPABLE PROPERTIES *source\_property drain\_property*

Used only in Calibre nmLVS-H and PERC.

### Parameters

- ***source\_property***  
Required name of a MOS Device element source pin property.
- ***drain\_property***  
Required name of a MOS Device element drain pin property.

### Description

Specifies the names of user-defined MOS Device properties for source and drain pins that can be swapped. MOS equivalent devices defined by LVS Device Type are included.

This statement is used in conjunction with LVS Push Devices using the SEPARATE PROPERTIES keyword. For MOS devices that have the specified properties and the source and drain pins are swapped during pushdown, the specified properties are written to the SEPARATE PROPERTIES file as swapped. This behavior is in addition to the default swapping of AS/AD and PS/PD properties in the same situation.

This statement may be specified more than once, and each pairing of *source\_property* and *drain\_property* must be unique in the rule file.

### Examples

This example shows a typical usage:

```
DEVICE MP gate gate(G) sd(S) sd(D) nw(B) [
    PROPERTY L, W, AS, AD, PS, PD, SCC, DCC
    ...
]
LVS PUSH DEVICES YES
LVS PUSH DEVICES SEPARATE PROPERTIES out.pdsp
// AS, AD, PS, PD are swappable by default
LVS MOS SWAPPABLE PROPERTIES SCC DCC // allow these to be swapped as well
```

## LVS Netlist All Texted Pins

Specification statement

### LVS NETLIST ALL TEXTED PINS {NO | YES}

Used only in Calibre nmLVS-H and PERC.

#### Parameters

- **NO**

Keyword that specifies that internally-floating pins are filtered out of the extracted layout netlist, whether they are texted or not. This is the default behavior.

- **YES**

Keyword that specifies that all texted cell pins should appear in the netlist; specifically, internally-floating pins should not be filtered out if they are texted.

#### Description

Controls filtering of texted cell pins in the Calibre nmLVS-H netlist extractor. In this context, a cell pin is texted if the respective net is texted in the cell, or if the respective net is connected to texted port objects in the cell (at the top level of the cell, excluding the sub-hierarchy of the cell).

Internally-floating pins are pins of a cell that are not connected to devices, or to [LVS Box](#) cell placements within the cell or within the sub-hierarchy of the cell. Such pins are normally omitted from the .SUBCKT definition of the cell and from subcircuit calls (X statements) referencing the cell, whether they are texted or not. Specifying YES causes internally-floating, texted pins to appear in the extracted netlist.

Texted port objects can be created with [Port Layer Text](#) specification statements. To qualify, the port names must be non-null. Note that unnamed port objects, such as those created by [Port Layer Polygon](#), do not qualify.

This statement may be specified at most once.

#### Examples

```
// Do not netlist internally-floating pins.  
LVS NETLIST ALL TEXTED PINS NO
```

# LVS Netlist Allow Inconsistent Model

Specification statement

## LVS NETLIST ALLOW INCONSISTENT MODEL {NO | YES}

### Parameters

- **NO**

Keyword that causes LVS to represent certain MOS devices (discussed later in this section) in the extracted netlist with subcircuit calls and to generate primitive subcircuit definitions, as needed. This is the default behavior.

- **YES**

Keyword that causes LVS to represent certain MOS devices (discussed later in this section) in the extracted netlist as M device elements. Do not use this option unless you want to restore deprecated functionality.

### Description

This statement is provided for backward compatibility with deprecated functionality. This statement controls how certain MOS devices are represented in the extracted netlist. MOS devices that do not follow the prescribed conventions regarding model names are affected. Specifically, these [Device](#) elements are affected:

- Device MN with model names not beginning with N.
- Device MP with model names not beginning with P.
- Device ME with model names not beginning with E.
- Device MD with model names not beginning with D.
- Device M with model names not beginning with N, P, E, or D.

Case (upper- or lowercase) is immaterial.

The (default) NO setting instructs LVS to represent these devices in the netlist with subcircuit calls, and to generate primitive subcircuit definitions as needed.

A YES setting instructs LVS to represent these devices with M elements in the extracted layout netlist (assuming that they satisfy the necessary conditions with respect to number of pins, pin names, and so forth). Selecting YES can cause unexpected behavior in other specification statements, such as [Trace Property](#).

For example, consider these Device operations:

```
DEVICE MP(AAA) ... // MP, model name starts with 'A'.
DEVICE M(PMOS) ... // M, model name starts with 'P'.
```

## LVS Netlist Allow Inconsistent Model

---

If you specify YES, these devices appear in the extracted layout netlist as shown next. The resulting LVS component types and subtypes are shown in parentheses.

```
M1 1 2 3 4 AAA    (LVS component type M, subtype AAA)
M2 1 2 3 4 PMOS   (LVS component type MP, subtype PMOS)
```

Notice that the LVS component types are different from the original element names.

If you specify NO, these devices would appear in the netlist as follows:

```
.SUBCKT MP g s d b
.ENDS

.SUBCKT M g s d b
.ENDS

X1 2 3 1 4 MP $[AAA] (LVS component type MP, subtype AAA)
X2 2 3 1 4 M $[PMOS] (LVS component type M, subtype PMOS)
```

This change ensures that LVS specification statements are processed as expected.

Consider this example:

```
DEVICE MP(AAA) ...
TRACE PROPERTY MP(AAA) ...
```

If you specify YES, the MP device would have LVS component type M in circuit comparison, and the Trace Property statement is not applied. If you specify NO, Trace Property is applied as expected.

Note that the element name/model name combinations listed previously are discouraged. Rather than specifying YES, you should consider changing your rule file Device statement so that they do not cause such problems.

## Examples

```
// Netlist inconsistent model devices as subcircuits.
LVS NETLIST ALLOW INCONSISTENT MODEL NO
```

## LVS Netlist Box Contents

Specification statement

### LVS NETLIST BOX CONTENTS {YES | NO}

Used only in Calibre nmLVS-H, PERC, and xRC.

#### Parameters

- **YES**

Keyword that instructs the hierarchical circuit netlister to write the internal contents of layout LVS Box cells to the output netlist. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs the hierarchical circuit netlister to output layout LVS Box cells only by their respective .SUBCKT and .ENDS statements and by comment lines.

#### Description

Specifies whether the Calibre nmLVS-H SPICE netlister includes the internal contents of LVS Box cells in the output netlist. This statement can be specified once.

If you specify YES, or if this statement is omitted, then LVS Box cells appear in the netlist with full contents just like regular cells.

If you specify NO, then layout LVS Box cells are described in the output netlist only by their respective .SUBCKT and .ENDS statements, and by comment lines; SPICE elements or subcircuit calls within the LVS Box subcircuit are not written out. This is indicated with the words CONTENTS DISCARDED in a comment line following the .SUBCKT statement, as shown in this example:

```
.SUBCKT inv GROUND VCC OUT
** N=4 EP=3 IP=0 FDC=2 CONTENTS DISCARDED
.ENDS
```

Note that this statement does not affect the subcircuit definitions of layout cells that are not LVS Box cells, even if those cells are placed exclusively within LVS Box cells. Such cells are fully described in the netlist, regardless of this statement.

The [LVS Preserve Box Cells](#) YES statement has the same effect as LVS Netlist Box Contents NO and supersedes LVS Netlist Box Contents YES. Additionally, LVS Preserve Box Cells YES disables seed promotion out of box cells, so the devices that originate from box cells are not netlisted. LVS Netlist Box Contents NO does not have this latter effect.

#### Examples

```
// Netlist the internal circuitry of LVS BOX cells.
LVS NETLIST BOX CONTENTS YES
```

# LVS Netlist Comment Coded Properties

Specification statement

## LVS NETLIST COMMENT CODED PROPERTIES {YES | NO}

Used only in Calibre nmLVS-H and PERC.

### Parameters

- **YES**

Keyword that indicates the Calibre nmLVS-H circuit netlister should use the following comment-coded properties:

- For capacitors (element C and equivalent devices), property A is netlisted as \$A and property P is netlisted as \$P.
- For bipolar transistors (element Q and equivalent devices), property L is netlisted as \$L and property W is netlisted as \$W.

This is the default.

- **NO**

Keyword that causes capacitors and bipolar transistors to be netlisted with regular device parameters. The device parameters A and P for capacitors, and parameters L and W for bipolar transistors, are not coded as comments.

### Description

Specifies whether the Calibre nmLVS-H circuit netlister, invoked by calibre -spice or the Query Server, should use the comment-coded parameters for capacitors and bipolar transistors. This statement only affects how such properties get netlisted, not whether such properties are calculated. Properties A and P for capacitors are not calculated by default, so they would have to appear in a user-defined property computation program. Similarly, the L and W properties for bipolar devices are not calculated by default.

Calibre-specific device annotations, such as \$X, \$Y, \$T, and \$D, are always netlisted as comments.

### Examples

Assume a Device statement like this:

```
DEV C cap_layer anode(POS) cathode(NEG)
[
    property C, A, P
    CA = 1E-02 // area cap
    CP = 1E-10 // perim cap
    CCA = CA * (unit_cap() / (unit_len() * unit_len()))
    CCP = CP * (unit_cap() / unit_len())
    A = area(cap_layer)
    P = perim(cap_layer)
    C = CCA * A + CCP * P
]
```

This could extract a device instance like this when LVS Netlist Comment Coded Properties YES is used:

```
C0 1 10 8.26001e-13 $A=8.26001e-11 $P=3.6358e-05 $X=-22206 $Y=62402 $D=1
```

## LVS Netlist Comment Coded Substrate

Specification statement

### LVS NETLIST COMMENT CODED SUBSTRATE {YES | NO}

Used only in Calibre nmLVS-H and PERC.

#### Parameters

- **YES**

Keyword that indicates the Calibre nmLVS-H circuit netlister should use the comment-coded parameter \$SUB to represent substrate pins in 3-pin diodes, 3-pin capacitors, and 3-pin resistors (standard SPICE elements D, C, and R, respectively). This is the default if this statement is not specified.

- **NO**

Keyword that indicates 3-pin diodes, 3-pin capacitors, and 3-pin resistors are represented as subcircuit calls (X elements) in the netlist.

#### Description

Specifies whether the Calibre nmLVS-H circuit netlister (for example, in calibre -spice) should use the comment-coded parameter \$SUB to represent substrate pins in 3-pin diodes, 3-pin capacitors, and 3-pin resistors. If you specify NO, then such devices are represented in the netlist with subcircuit calls (X elements), and respective empty .SUBCKT definitions are written to the netlist as well (unless disabled by other statements).

For example, with the default YES setting, a 3-pin resistor is represented as follows:

```
R 1 2 50 $SUB=3 $X=10000 $Y=20000
```

When you specify NO, that resistor is represented by an X subcircuit call with a respective empty R subcircuit definition:

```
.SUBCKT r pos neg sub  
.ENDS
```

```
X0 1 2 3 r r=50 $X=10000 $Y=20000
```

This statement also controls Query Server netlisting commands in a similar manner, as described in the section “[Customized SPICE Netlist Format](#)” of the *Calibre Query Server Manual*.

This statement may appear once in your rule file.

#### Examples

```
// Netlist comment-coded $SUB parameter for C, D, and R devices  
LVS NETLIST COMMENT CODED SUBSTRATE YES
```

# LVS Netlist Unnamed Box Pins

Specification statement

## LVS NETLIST UNNAMED BOX PINS {YES | NO}

Used only in Calibre nmLVS-H, PERC, and xRC.

### Parameters

- **YES**

Keyword that instructs nmLVS-H to include unnamed pins that are connected to devices inside **LVS Box** cells. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs nmLVS-H to omit all unnamed pins from .SUBCKT definitions and subcircuit calls (X statements) of LVS Box cells in the layout set.

### Description

Specifies whether the hierarchical SPICE netlister includes certain unnamed pins of LVS Box cells in the output netlist.

If NO is specified, then all unnamed pins are omitted from .SUBCKT definitions and from subcircuit calls (X statements) of LVS Box cells in the layout set.

If YES is specified, or if this statement is omitted, then unnamed pins appear in the netlist if they are connected to devices or LVS Box placements inside the box cell (including sub-hierarchy of the cell), or if they are connected to texted port objects in the box cell (at the top level of the box cell, not including sub-hierarchy of the cell).

Texted port objects can be created with the [Port Layer Text](#) specification statement. Unnamed pins of LVS Box cells that are not connected in any of these ways are always omitted, regardless of this statement.

Named pins of LVS Box cells are always included in the netlist. Recall that a pin name is simply the name of the respective net in the cell. A pin is named if the respective net is named (texted).

This statement can be specified once in your rule file.

### Examples

```
// Netlist unnamed pins in LVS Box cells.
LVS NETLIST UNNAMED BOX PINS YES
```

## LVS NL Pin Locations

Specification statement

### LVS NL PIN LOCATIONS {NO | YES}

Used only in flat Calibre nmLVS and PERC.

#### Parameters

- **NO**  
Keyword that instructs LVS not to report pin locations and related information for user-defined devices in the netlist created when you specify the -nl switch on the command line.
- **YES**  
Keyword that instructs the tool to report pin locations and related information for user-defined devices in the netlist created when the -nl switch is specified in the command line.

#### Description

Controls the layout netlist created by the -nl command line option of flat Calibre nmLVS (but not ICtrace). It instructs flat LVS to report pin locations and related information for user-defined devices in the -nl netlist.

When you specify YES for this statement in the rule file, each user-defined device operation of the form:

```
DEVICE userdev(usertype) seed pin_layer_1(pin1)pin_layer_2(pin2) ...
```

causes the -nl netlist to contain a device template of the form:

```
*.DEVTMPLT <n> userdev(usertype) seed pin_layer_1(pin1) pin_layer_2(pin2)  
...
```

where <n> is a non-negative integer between 0 and the number of [Device](#) operations in the rule file; it serves as a unique identifier of the device template. Individual user-defined device instances are then represented as shown in the following example:

```
x1 1 2 userdev $[usertype] $dt=n  
+$PIN_XY=7.85,-48.1,12.85,-48.1 ...
```

where \$dt=n is the DEVTMPLT identifier <n> and \$PIN\_XY contains x,y coordinates for each pin.

This statement can be specified once in your rule file.

#### Examples

```
/* Do not netlist pin locations for user-defined devices when using -nl  
option. */  
LVS NL PIN LOCATIONS NO
```

## LVS Non User Name

Specification statement

### **LVS NON USER NAME {INSTANCE | NET | PORT}**

*regular\_expression* [*regular\_expression* ...]

#### Parameters

- **INSTANCE**

Keyword that specifies instance names should not be treated as user-given names in LVS.

- **NET**

Keyword that specifies net names should not be treated as user-given names in LVS. This does not apply to net names designated as power or ground names.

- **PORT**

Keyword which specifies that port names should not be treated as user-given names in LVS. This does not apply to port names designated as power or ground names.

- ***regular\_expression***

A required list of one or more regular expressions that determine the names that are not to be treated as user-given. Regular expression syntax is discussed under “[Layout Rename Text](#)” on page 787. This statement uses only the GNU implementation of extended regular expression syntax. You should consult your local UNIX man page covering regular expressions for details.

#### Description

Specifies instance, net, or port names that should not be treated as user-given names in LVS, even if they otherwise fit the criteria for user-given names. User-given names form initial correspondence points in LVS; non-user-given names do not.

One of the keywords INSTANCE, NET or PORT must be specified first; they indicate that the statement operates on instance names, net names, or port names, respectively. (The selection of INSTANCE, NET, or PORT can be specified with a string variable.) This is followed by a list of one or more regular expressions.

Any name of the appropriate type (instance, net, or port, respectively) that contains a substring matching one or more of the regular expressions, is not considered a user-given name in LVS. For example, the name GND matches the regular expression N because GND contains the substring N; therefore GND would be considered a non-user-given name.

Regular expressions are accumulated across statements, which means that order of statements is important. For example, statements of the form LVS Non User Name NET are cumulative, and the later statements of this form in the rule file are affected by earlier ones.

If a net or port name is normally considered user-given, and it is designated as power or ground name, then it will continue to be user-given even if it matches a regular expression that appears in a LVS Non User Name statement. (Names are designated as power or ground names with the

[LVS Power Name](#) and [LVS Ground Name](#) specification statements or, in Pyxis Layout, with the \$set\_lvs\_power\_names() and \$set\_lvs\_ground\_names() commands).

As is the case with names in general, regular expressions that contain SVRF special characters must be in quotes. It is not mandatory to quote other regular expressions. However, quoting all regular expressions is a good practice.

String matching is case-sensitive if [LVS Compare Case NAMES](#) or LVS Compare Case YES is specified, and is case-insensitive otherwise.

In hierarchical LVS, this statement with the PORTS keyword operates on hcell ports as well as top-level ports. Recall that hcell ports are processed even when [LVS Ignore Ports YES](#) is specified.

This statement may appear any number of times. LVS Non User Name settings are reported in the LVS Setup section of the LVS report.

### Examples

#### Example 1

The following example specifies that net names consisting of the letter n and followed by zero or more digits are not user-given. For example, n1, n123, and n are not user-given, but nn1 and n12a are user-given.

```
LVS NON USER NAME NET "^\n[0-9]*$"
```

#### Example 2

The following example specifies that net names consisting of the letter n and followed by one or more digits are not user-given. For example, n1 and n123 are not user-given, but n, nn1 and n12a are user-given.

```
LVS NON USER NAME NET "^\n[0-9]+$"
```

#### Example 3

The following example specifies that net names beginning with the strings nn or mm are not user-given. For example, nn1 and nnabc are not user given.

```
LVS NON USER NAME NET "^\nn" "^\mm"
```

#### Example 4

The following example specifies that instance names containing the letter a are not user-given.

```
LVS NON USER NAME INSTANCE "a"
```

#### Example 5

The following example specifies that instance names and net names should never be treated as user-given. Only port names may be treated as user-given. The regular expression “.” matches any single character, causing any name to match this regular expression since any name contains a substring consisting of some single character.

```
LVS NON USER NAME INSTANCE "."
LVS NON USER NAME NET ".."
```

**Example 6**

The following example specifies that port names containing one or more underscore are not user-given.

```
LVS NON USER NAME PORT "_+"
```

## LVS Out Of Range Exclude Zero

Specification statement

### LVS OUT OF RANGE EXCLUDE ZERO {NO | YES}

#### Parameters

- **NO**

A required keyword that specifies out-of-range Trace Property absolute tolerance warnings will be issued for property values equal to 0. This is the default behavior if you do not specify this statement.

- **YES**

A required keyword that specifies out-of-range Trace Property absolute tolerance warnings will not be issued for property values equal to 0.

#### Description

This statement only applies to cases when Trace Property ... ABSOLUTE is specified. This statement controls whether out-of-range warnings are issued for property values of 0. The range check is performed to warn about cases in which the units and scale of the absolute tolerance do not correspond to the units and scale of the property being checked.

This statement may be specified once.

When tracing properties with ABSOLUTE tolerance, LVS performs a preliminary range check and issues a warning if property values are significantly smaller than the specified tolerance. See the Trace Property [Built-in Property Classification](#) section for a description of this behavior. The warning reads as follows:

```
*****
ABSOLUTE TRACE PROPERTY TOLERANCES OUT OF RANGE
*****
```

(In the following TRACE PROPERTY rules, most values found for the respective property in the source were significantly smaller than the ABSOLUTE tolerance value specified in the rule. The tolerance value was probably specified with the wrong scale or units. Note: only matched instances are counted).

```
TRACE PROPERTY mp l l
TRACE PROPERTY mp w w
```

If you specify NO (the default), out-of-range warnings are issued for all values of out-of-range properties, including 0.

If you specify YES, property values that are exactly 0 are ignored for the purposes of the warning described previously; however, property value comparison still occurs for values of 0. The intent of the YES option is to defeat the out-of-range warning when property values of 0 are intentional and are significantly below the tolerance value.

## Examples

Assume the following is specified in the rule file:

```
TRACE PROPERTY V dc dc 0.1 ABSOLUTE
// dc is expected to match within an absolute value of 0.1
```

### Example 1

```
LVS OUT OF RANGE EXCLUDE ZERO NO
// default; dc property values equal to 0 can cause a tolerance
// out-of-range warning.
```

### Example 2

```
LVS OUT OF RANGE EXCLUDE ZERO YES
// dc property values equal to 0 do not cause a tolerance
// out-of-range warning.
// the property value comparison is still performed.
```

## LVS Pin Name Property

Specification statement

**LVS PIN NAME PROPERTY** *name* [*name* ...]

Used only in ICtrace.

### Parameters

- *name*

A required name of a property. You can specify *name* any number of times in this statement. The *name* can be a string variable.

### Description

Specifies a prioritized list of zero or more property names to be used in determining pin names for source instance pins in ICtrace. This statement can be specified any number of times. The default is no statement, but “pin” is always implied as the last property name in the list.

In Pyxis Layout, this statement sets the value of application variable lvs\_pin\_name\_property.

### Examples

```
/* Use this property name for determining instance pin names in the
source. */
LVS PIN NAME PROPERTY phy_pin
```

# LVS Power Name

Specification statement

**LVS POWER NAME** *name* [*name* ...]

## Parameters

- *name*

A required name of a power net. You can specify *name* any number of times in this statement, and the nets are independent of one another. The string *name* can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. The *name* can be a string variable.

## Description

Specifies a list of one or more independent power net names. Power net names are used by LVS in logic gate recognition, in filtering of unused MOS transistors, and in power supply verification in Direct mode ICtrace. This statement can appear multiple times in a rule file.

No power names are assumed by default, so this statement must be used to declare any power nets.

The connectivity extractor uses power and ground names to resolve text conflicts (short circuits). Conflicts in which different names are found on a single net are resolved in order of precedence as follows:

1. Power names, if specified, in rule file order.
2. Ground names, if specified, in rule file order.
3. Alphabetically least name.

In addition, the hierarchical connectivity extractor resolves net name conflicts during high-short resolution in the same manner.

Power net names are used in [Pathchk](#) and [ERC Pathchk](#) operations for path tracing. They are also used for filtering of unused devices in [Device Layer](#), Pathchk, and ERC Pathchk operations.

The “[Power and Ground Nets](#)” section of the *Calibre Verification User’s Manual* discusses how such nets are used in LVS comparison.

A power name is considered badly-formed if it is classified as a non-user-given net name by LVS criteria in both layout and source. Such names should not appear in LVS Power Name statements or related Pyxis Layout variable settings. The section “[User-Given Names](#)” in the *Calibre Verification User’s Manual* discusses user-given (as opposed to system-generated) net names.

You can specify power supply management options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Setting Power Supply Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In Pyxis Layout, this statement sets the value of application variable lvs\_power\_names.

## LVS Power Name

---

See also [LVS Ground Name](#), [LVS Recognize Gates](#), [Pathchk](#), [LVS Cell Supply](#), [LVS Filter Unused MOS](#), and [LVS Filter Unused Option](#).

### Examples

```
//VDD and VCC are not virtually connected; use these as power net names.  
LVS POWER NAME VDD VCC
```

# LVS Precise Interaction

Specification statement

## LVS PRECISE INTERACTION {YES | NO}

Used only in Calibre nmLVS-H, PERC, and xRC.

### Parameters

- **YES**  
Keyword that instructs the hierarchical device recognition process to compute precise interaction regions for layers involved in [Device](#) operations. This is the default behavior.
- **NO**  
Keyword that instructs the hierarchical device recognition process not to compute precise interaction regions.

### Description

Precise interaction region computation is an algorithm in hierarchical device recognition that is designed to minimize seed promotion. This computation can be excessively long in some designs (although this is not common); the time consumed appears as DEVICE RECOGNITION TOTAL PRECISE INTERACTION TIME in the Calibre transcript.

If this computation is excessively time-consuming, you can disable it by specifying NO. This may cause more initial seed promotion to occur than would normally be the case. However, with [LVS Push Devices](#) YES specified (the default), Calibre should be able to recover from much of the resulting seed promotion, so disabling precise interaction computation is relatively safe.

However, even with LVS Push Devices YES specified, seed promotion does occur initially; so LVS Precise Interaction NO should be used with care. Do not use LVS Precise Interaction NO together with LVS Push Devices NO, or with the [Mask SVDB Directory](#) options that disable device pushdown: PINLOC, or CCI without NOPINLOC.

This statement may be specified once.

### Examples

```
// Disable Precise Interaction Reporting
LVS PRECISE INTERACTION NO
LVS PUSH DEVICES YES
```

## LVS Preserve Box Cells

Specification statement

### LVS PRESERVE BOX CELLS {NO | YES}

Used only in Calibre nmLVS-H and PERC.

#### Parameters

- **NO**

Keyword that instructs the tool to allow seed promotion out of LVS Box cells. Any devices that originate from seed promotion out of box cells are written to the extracted netlist. This is the default behavior.

- **YES**

Keyword that instructs the tool to disable device formation due to seed promotion out of LVS Box cells. Such devices are not written to the extracted netlist. This keyword also causes LVS Box cells to be written as empty subcircuits in the netlist.

#### Description

Controls the writing of LVS Box cell contents during layout netlist extraction. This statement may be specified once.

If NO is specified (the default), then devices can be formed by seed promotion out of box cells. Such devices are written in the extracted netlist.

If YES is specified, then the contents of box cells are not written to the layout netlist. Instead, these cells are written out as empty subcircuits. This behavior is identical to [LVS Netlist Box Contents NO](#). (LVS Preserve Box Cells YES supersedes LVS Netlist Box Contents YES, which is the default.)

In addition, when YES is specified, devices are not formed by seed promotion out of box cells. Note that this option does not prevent the hierarchical database engine from promoting geometries, including device seed shapes, from box cells. (Selective promotion of layout shapes by various layer operations is insensitive to box cells.)

When YES is specified, box cells in the netlist are annotated with comments: CONTENTS DISCARDED and DEVICE PROMOTION DISABLED appear in the first comment line after the net and pin counts. In addition, \*.PBOX appears in every subcircuit representing a preserved box cell. Here is an example of a LVS Box cell whose contents were discarded by this statement:

```
.SUBCKT inv GROUND VCC OUT
** N=4 EP=3 IP=0 FDC=2 CONTENTS DISCARDED DEVICE PROMOTION DISABLED
*.PBOX
.ENDS
```

This statement can be used in combination with [LVS Preserve Box Ports YES](#) to prevent nmLVS-H from modifying box cells to the greatest possible degree.

## Examples

This example shows how to preserve LVS Box cells in their original forms to the maximum degree possible.

```
// Preserve LVS BOX cells in their original forms as much as possible  
LVS PRESERVE BOX CELLS YES  
LVS PRESERVE BOX PORTS YES
```

## LVS Preserve Box Ports

Specification statement

### LVS PRESERVE BOX PORTS {NO | YES}

Used only in Calibre nmLVS-H and PERC.

#### Parameters

- **NO**

Keyword that instructs the tool to combine both named and unnamed ports of LVS Box cells during high-short resolution whenever possible. This is the default behavior.

- **YES**

Keyword that instructs the tool not to combine both named and unnamed ports of LVS Box cells during high-short resolution. This only applies when there is a single user-given name for some ports and the others are unnamed. The high-shorted ports with user-given names are combined, and the unnamed ports are combined separately.

#### Description

Controls how nmLVS-H handles named and unnamed ports of LVS Box cells during high-short resolution. For complete details of this process, see “[High-Short Resolution](#)” in the *Calibre Verification User’s Manual*. This statement applies during connectivity extraction and netlist comparison. It may be specified once.

When NO is specified (the default), high-short resolution combines named and unnamed ports of LVS Box cells. When YES is specified, high-short resolution does not combine such ports together in cases where there is a single user-given name for some ports and the others are unnamed. The ports with the user given name are combined separately from the unnamed ports. [Table 4-25](#) shows examples.

**Table 4-25. High-Shorted LVS Box Port Resolution Examples**

Original high-shorted box ports	High-short resolution with NO specified	High-short resolution with YES specified
A 2	A	A 2
A A 2 3 4	A	A 2
A B	A B	same as NO
A A A B B	A B	same as NO
A B 3	A B 3	same as NO
A A A B B 3 4 5	A B 3	same as NO
1 1 2 2	1	same as NO

The hierarchical connectivity extraction module enforces these conventions in the layout netlist.

The LVS comparison module performs its own high-short resolution equally in layout and source in order to bring both to a common representation. The resolution of high-shorted ports in LVS Box cells is consistent between the connectivity extraction and LVS comparison modules.

LVS Preserve Box Ports YES is frequently used with [LVS Preserve Box Cells](#) YES to prevent box cell modification to the greatest possible degree.

### Examples

```
// Allow high-shorted LVS Box port names to be combined  
LVS PRESERVE BOX PORTS NO
```

## LVS Preserve Floating Top Nets

Specification statement

### LVS PRESERVE FLOATING TOP NETS {NO | YES}

Used only in Calibre nmLVS-H and PERC.

#### Parameters

- **NO**

Keyword that instructs the tool not to preserve floating named nets (no device connected) in the top cell. This is the default behavior if you do not include this statement in your rule file, and causes only nets connected to devices to be written to the SPICE netlist.

- **YES**

Keyword that instructs the tool to preserve floating named nets in the top cell.

#### Description

Controls preservation of floating named top-level nets in the extracted layout SPICE netlist. If NO is specified, or the statement is not used, only top-level nets connected to devices are written to the SPICE netlist. If YES is specified, top-level nets that have names but no device attached are preserved in the SPICE netlist. These nets are represented with the \*.J SPICE statement (a shortcut for a \*.CONNECT statement) using identical parameters:

```
*.J <name> <name>
```

For example:

```
*.J net1 net1
```

Note that specifying identical parameters in a \*.J (or a \*.CONNECT) statement does not follow typical usage for such statements, but is used in this case to preserve the floating net name.

#### Examples

```
// Preserve floating top nets in the SPICE netlist
// This flags floating top-level nets during LVS comparison
LVS PRESERVE FLOATING TOP NETS YES
```

# LVS Preserve Parameterized Cells

Specification statement

## LVS PRESERVE PARAMETERIZED CELLS {NO | YES}

Used only in Calibre nmLVS-H and PERC.

### Parameters

- **NO**

Keyword that instructs the tool not to preserve the hierarchy of parameterized subcircuits. This is the default behavior if you do not include this statement in your rule file, and this setting causes flattening of parameterized subcircuits.

- **YES**

Keyword that instructs the tool to preserve the hierarchy parameterized subcircuits.

### Description

Controls flattening of parameterized subcircuits in nmLVS-H. A subcircuit is parameterized if it contains at least one subcircuit call that references it, and that call specifies parameter values.

If YES is specified, parameterized subcircuits are preserved, which means that cells are not flattened. The default is NO, which means that parameters are passed from subcircuit calls and cells are flattened because of this.

---

**Note**



Be aware that specifying YES causes parameters not to be passed from subcircuit calls to subcircuits.

---

Flattened, parameterized subcircuits do not serve as hierarchical entities in nmLVS-H; it ignores any references to them in hcell files, **Hcell** statements, or implicitly with the -automatch option (-automatch is generally discouraged). A warning is issued if flattened parameterized subcircuits appear in hcell files. This statement does not affect connectivity extraction.

### Examples

```
// Do not preserve the hierarchy of parameterized subcircuits.
LVS PRESERVE PARAMETERIZED CELLS NO
```

## LVS Property Initialize

Specification statement

**LVS PROPERTY INITIALIZE** *component\_type* [(*component\_subtype*)]  
‘[*lvs\_property\_initialize\_program*]’

### Parameters

- ***component\_type***  
A required LVS component type to which this statement applies.
- **(*component\_subtype*)**  
An optional component subtype of the ***component\_type*** parameter. If specified the ***component\_subtype*** must appear within parentheses. If ***component\_subtype*** is not present, then the LVS Property Initialize statement applies to all instances of the specified ***component\_type*** regardless of their subtype, except for subtypes that have their own LVS Property Initialize statement (that is, a statement with the same component type and with the component subtype of the instance).
- **‘[*lvs\_property\_initialize\_program*]’**  
A required device property initialization program, which appears between literal brackets. The complete description of this language appears under “[LVS Property Initialize Built-In Language](#)” on page 1867.

### Description

Instructs LVS comparison to add numeric properties simultaneously to instances in the layout and source input databases and to initialize those properties to specified values. This statement is not designed as a general property insertion mechanism, and it does not apply to netlist extraction.



#### Note

You should avoid using this statement except in special circumstances where your source netlist is incomplete or where consistency checking (see “[Reduction Program Semantic Checking](#)” on page 1864) of reduced device properties needs to be disabled.

---

The LVS Property Initialize statement includes a program written in a language similar to that used for device reduction effective property computation programs. The LVS Property Initialize programming language is somewhat more restricted than the effective property computation language.

When instances in the input databases contain property values, those values are made accessible to the ***lvs\_property\_initialize\_program***. Instances can be checked for the existence of property values, as well as what the property values are. The ***lvs\_property\_initialize\_program*** can then manipulate the property values.

Each property added and initialized by LVS Property Initialize is called a *generated* property. That is, such properties are not expected to be present in the input databases. Instead, the

properties are created by their appearance in the LVS Property Initialize statement, and are given the value computed in the *lvs\_property\_initialize\_program* for every instance read from input databases.

Specifying LVS Property Initialize in the rule file disables all consistency checks related to checking effective device properties (from any of the LVS Reduce family of statements) against **Device** definitions, regardless of device type, subtype, or property name. For example, the statement:

```
LVS PROPERTY INITIALIZE MN [ PROPERTY per per = 1 ]
```

will disable all consistency checks regarding any property in any device.

## Examples

### Example 1

This example shows the initialization of three properties in layout and source. It also disables all consistency checking for reduction of all device properties specified in the rule file.

```
LVS PROPERTY INITIALIZE type(subtype) [
    PROPERTY eqpar, eqser, sq2
    eqpar=1
    eqser=1
    sq2 = SQRT(2)/2
]
```

### Example 2

The following example checks MN instances for the existence of property w. If the w property is not found, then w is initialized to 0; otherwise, the w value is maintained. Because w is declared as a property, w must have a value after the user program executes; therefore, the IF statement must guarantee w gets a value from either branch of the conditional block.

```
LVS PROPERTY INITIALIZE MN [
    PROPoerty w
    input_w = INPut_NUMeric_VALue ( w ) // get w if present for instances
    IF ( IS_MISSING ( input_w ) == 1 ) { // check to see if w is present
        w = 0 // w is missing, initialize it to 0
    } ELSE {
        w = input_w // w is present, so use the input value
    }
]
```

## LVS Property Map

Specification statement

### LVS PROPERTY MAP *component\_type target\_property database\_property*

[‘*spice\_parameter*’] {LAYOUT | SOURCE}

#### Parameters

- ***component\_type***

A required LVS component type to which this statement applies.

- ***target\_property***

A required property name that appears in an applicable rule file construct (see the Description section for the statements and constructs to which property maps apply). LVS represents this property with the value of ***database\_property***.

- ***database\_property***

A required property name in the input database to which the property map applies. LVS uses this value to represent the ***target\_property*** in the rule file. The ***database\_property*** must be a character string.

- **(*spice\_parameter*)**

An optional name of a SPICE parameter (a letter), which must be enclosed in parentheses, to be extracted from values of ***database\_property***. The SPICE parameter can be uppercase or lowercase.

- **LAYOUT**

Keyword that specifies this statement applies only to instances in the layout.

- **SOURCE**

Keyword that specifies this statement applies only to instances in the schematic.

#### Description

Instructs LVS to map property names in certain LVS specification statements to property names in the specified input database. This statement indicates that for instances of the specified ***component\_type***, the ***target\_property*** is represented in the input database with the ***database\_property***. For example:

```
LVS PROPERTY MAP C "AREA" "A" SOURCE
```

specifies that the AREA property somewhere in the rule file is represented by property “A” in the source database for C-type devices.

This statement only applies to numeric and string property names in these cases:

- The TOLERANCE or TOLERANCE STRING specifications of the LVS Reduce family of statements.
- The *effective\_property\_computation* block of the LVS Reduce family of statements.

- The *trace\_property\_computation* block of a [Trace Property](#) statement.
- The *lvs\_property\_initialize\_program* of the [LVS Property Initialize](#) statement.

TOLERANCE or TOLERANCE STRING statements and effective property computation formulas should be expressed in terms of *target\_property*. In effective property computation, the result is mapped back to the *database\_property*, so the resulting reduced device has a property with the original *database\_property* name and the computed effective value.

The mapping specified by this statement must be a one-to-one correspondence for a given component type and input database. Specifically, for a given component type and input database, you cannot indicate different input properties for the same target property or different target properties for the same input property.

Property tracing should occur based upon property names in the netlists, not *target\_property* names.

This statement does not apply to the LVS Filter family of statements, [LVS Split Gate Ratio](#), or [Trace Property](#) statements that have no property computation section.

This statement can be specified any number of times but duplicates are not allowed.

Under normal circumstances, this statement is not used when comparing a geometric layout to SPICE in Calibre.

In ICVerify, this statement is used with input databases that have flexible property naming conventions, such as EDDM or ICtrace(D) layout. This statement is applicable when the input databases use different property names or when SPICE-like syntax, such as instpar(w), is used.

## Examples

### Example 1

The following statement specifies that, for MN devices, the property W is represented as WIDTH from the input layout database.

```
LVS PROPERTY MAP MN W "WIDTH" LAYOUT
```

### Example 2

The following statement specifies that, for MN devices, the property W is encoded within the INSTPAR property in the input source database using the LVS SPICE-like property syntax.

```
LVS PROPERTY MAP MN W INSTPAR(W) SOURCE
```

### Example 3

In this example, MOS transistor width is represented with a property called WIDTH in the source database. However, in extracted layout devices, width is called W. You want to specify a TOLERANCE for parallel MOS reduction based on W values. To do that, use LVS Property Map statements in the source to map W to the WIDTH property.

```
DEVICE MP pgate pgate psd psd nwell
DEVICE MN ngate ngate nsd nsd psub

LVS PROPERTY MAP MP W "WIDTH" SOURCE
LVS PROPERTY MAP MN W "WIDTH" SOURCE
```

## LVS Property Map

---

```
LVS REDUCE PARALLEL MOS YES [  
    tolerance W 0  
    effective W  
    W = sum( W )  
]
```

# LVS Property Resolution Maximum

Specification statement

**LVS PROPERTY RESOLUTION MAXIMUM {*number* | ALL}**

## Parameters

- ***number***

A required non-negative integer that specifies the maximum size of ambiguity groups that LVS resolves using property values.

- **ALL**

Keyword that specifies unlimited elements within a group of ambiguous elements.

---

**Note**



Under most circumstances, the ALL keyword should not be used.

---

## Description

Specifies that LVS use property values to resolve all groups of ambiguous elements that contain no more than a maximum number of elements. This statement may be specified once in your rule file. Default: 32.

This rule generally applies to highly parallel and symmetric circuits where parts of the circuit can be interchanged without affecting connectivity. LVS uses property values to resolve ambiguities as a last resort. If a circuit contains more than the specified maximum number of ambiguous elements, LVS attempts resolution by arbitrarily matching them; this may cause LVS discrepancies.

If property resolution is needed during the LVS comparison, and the ***number*** specified in this statement is insufficient to ensure that all ambiguous nets or instances are processed by the ambiguity resolution, then the following warning is issued:

Warning: LVS property resolution maximum exceeded.

If this warning is issued, a simplified form of the ambiguity resolution is used for some nets or instances, which can result in false property errors.

Note that there is a potential performance penalty for specifying large property resolution maximum values. In particular, the ALL keyword should be avoided.

## Examples

The following tells LVS to use property values to resolve ambiguity groups of up to 200 instances:

```
LVS PROPERTY RESOLUTION MAXIMUM 200
```

## LVS Push Devices

Specification statement

Syntax 1:

**LVS PUSH DEVICES {{[YES] [LDE]} | NO}**

Syntax 2:

**LVS PUSH DEVICES {SEPARATE PROPERTIES {[filename] [AGF]} }**

Used only in Calibre nmLVS-H, PERC, and xRC.

### Summary

Controls the attempt by the hierarchical device recognition process to push devices back down the hierarchy after seed promotion.

### Parameters

- **YES**

Keyword that instructs the hierarchical device recognition process to attempt to push devices back down the hierarchy after seed promotion. This is the default for Syntax 1. The details for **YES** in Syntax 2 are discussed under the **SEPARATE PROPERTIES** keyword set.

- **LDE**

Keyword that instructs the hierarchical device recognition process to attempt to push devices back down the hierarchy after seed promotion when the devices are considered logically equivalent. Specifying this keyword triggers the **YES** keyword, whether **YES** is specified or not. See “[Logical Device Equivalence Pushdown](#)” for details.

- **NO**

Keyword that instructs the hierarchical device recognition process not to push promoted devices down the hierarchy. The details for **NO** in Syntax 2 are discussed under the **SEPARATE PROPERTIES** keyword set.

- **SEPARATE PROPERTIES {[filename] [AGF]}**

This keyword set is used only by flows where LVS is supporting pushdown backannotation. It is a required keyword set in Syntax 2.

The **SEPARATE PROPERTIES** keyword directs device pushdown to ignore certain properties when evaluating candidate devices for equivalence. The result is, only *LVS-actionable* properties prevent pushdown. All other properties are ignored when a device is evaluated for pushdown.

---

#### Note



This syntax is used in conjunction with the Calibre Connectivity Interface (CCI) for pushdown backannotation. See “[Customized Files for Pushdown Backannotation \(PDBA\)](#)” in the *Calibre Query Server Manual* for additional details.

---

The LVS Push Devices **SEPARATE PROPERTIES** statement is only valid if LVS Push Devices **YES** is specified in the rule file in a separate statement or is active by default. Also, the [Mask SVDB Directory](#) is required when **SEPARATE PROPERTIES** is specified without a *filename*. See “[SEPARATE PROPERTIES Usage](#).”

*filename* — Optional pathname of a separated annotation properties netlist. The specified file is generated during connectivity extraction.

**AGF** — Optional keyword that specifies the output netlist format matches the Calibre Connectivity Interface (CCI) LAYOUT NETLIST HIERARCHY AGF command’s format. Without this option, the generated *filename* matches the extracted netlist. See “[Customized Layout SPICE Netlists](#)” in the *Calibre Query Server Manual* for more information about CCI commands.

## Description

This statement comes in two forms. Syntax 1 specifies whether hierarchical device recognition should attempt to recover from *seed promotion* situations by pushing promoted devices back down the hierarchy. Syntax 2 is used in conjunction with the Calibre Connectivity Interface for pushdown backannotation in parasitic extraction flows.

Seed promotion is an internal process that occurs during hierarchical connectivity extraction (calibre -spice). Seed promotion is used to assist device classification in a hierarchical structure by promoting seed polygons up the hierarchy in order to form devices at a single level of the hierarchy. (Seed polygons are polygons on the **device\_layer** parameter in the [Device](#) operation.) Reversing this process is called *device pushdown*.

Device pushdown causes *equivalent* promoted devices to be pushed down the hierarchy as low as possible but never below the level where the fully-formed seed polygon resides. Devices that are equivalent typically have the same component type, subtype, properties, and property values. Such devices are candidates for pushdown, which occurs by default.

Note that seed promotion occurs initially even when **YES** is specified; promoted devices are initially recognized at a higher level of hierarchy and are pushed down later in a post-processing step. The initial seed promotion is reported as TENTATIVE SEED PROMOTIONS in the Calibre transcript; the final seed promotion (after application of device pushdown) is reported as SEED PROMOTIONS in the transcript.

The nmLVS hierarchical SPICE netlister places \*.SEEDPROM statements in all subcircuits that were subject to seed promotion.

The **NO** setting in Syntax 1 is provided for backward compatibility with releases prior to Calibre 2003.3, where promoted devices were not pushed down. Normally, this statement can be omitted from your rule file and the default **YES** setting is used.

Each form of the LVS Push Devices statement may be specified only once.

If the LDE option is specified, it is disabled with a warning if either of the [Mask SVDB Directory](#) options CCI or XRC are specified.

## Situations Preventing Pushdown

When **YES** is specified in Syntax 1, pushdown occurs only if it can be performed on a template-specific basis; it does not occur if the results would be instance-specific. In other words, a promoted device can be pushed back into a cell only if respective promoted devices were successfully formed above *all* placements of that cell, and all those devices conform to these conditions:

- They were promoted from the same respective seed polygon.
- They were formed by the same Device operation.
- They have similar pin connectivity.
- They have the same property values.

(The LDE option relaxes the second condition. See [Logical Device Equivalence Pushdown](#).)

For example, consider a cell placed 1000 times, containing a device seed that is promoted out of the cell by seed promotion. If all 1000 individual device instances are successfully formed up the hierarchy, they all have the same property values, and are formed by the same Device operation, then they can be pushed back into the cell. A single device is formed in the cell and the 1000 individual copies at the higher level of hierarchy are removed.

However, if only 900 of the individual device instances are actually formed up the hierarchy and the rest are classified as bad devices, then none of them are pushed down. Similarly, if 500 of the individual device instances are recognized by one Device operation and the rest are recognized by a different Device operation (for example, with a different number of pins), then none are pushed down. If property values differ among the individual device instances, then none of them are pushed down. Finally, if pins connect differently on any of the devices, then none of them are pushed down.

In addition, a promoted device cannot be pushed down from a parent cell into a target cell if *any* of the following are true:

- The device was promoted because of seed-to-seed interaction (that is, interaction between seed-layer polygons from different cells or different levels of hierarchy).
- The device has swappable pins, and some or all of the respective pin shapes are missing in the target cell.

Exception: If the swappable pins are consistently shorted together in the parent cell or cells, then the device can be pushed down.

- The device owns properties derived from swappable pins, and the target cell is placed more than once in the design.

A property is *derived* from a pin if its computation involves an operation that takes the pin name as an argument. For example, in the expression:

```
Z = perimeter(POS)
```

if POS is a pin name, then property Z is derived from pin POS. The following Device operations fall under this restriction.

```
DEVICE C cap1 m1 m2 (POS NEG) [
    property Z1, Z2
    Z1 = perimeter(POS)
    Z2 = perimeter(NEG)
]

DEVICE R res1 m1(POS) m1(NEG) [
    // POS and NEG implicitly swappable.
    property Z
    Z = area(POS) + area(NEG)
]
```

Note that the second Device operation could be re-written as follows, which would exempt it from this restriction:

```
DEVICE R res1 m1 m1 [
    property Z
    Z = area(m1)      // Pin layer used instead of pin names.
]
```

**Exception:** Built-in properties of MOS devices that are known to be swappable do not disqualify pushdown in this manner. These properties are AS, AD and PS, PD. LVS assumes that properties AS and PS belong to pin S and properties AD and PD belong to pin D. When pushing devices, LVS allows properties AS and AD to be swapped, and PS and PD to be swapped only if pins S and D are also swapped. In addition, properties W and L in MOS devices, and property R in resistor devices, do not disqualify pushdown in this manner. Such properties are assumed not to be tied to any particular pin. In this context, MOS devices are element names MP, MN, ME, MD, and M, and resistor devices are element name R.

If device pushdown is specified either explicitly or by default, device pushdown is not performed if pin location information must be generated (as in Calibre xRC flows). [Table 4-26](#) shows various situations that can occur.

**Table 4-26. Device Pushdown Behaviors**

Rule File Statements	Pushdown Occurs?
Mask SVDB Directory PINLOC	No
Mask SVDB Directory CCI	No
Mask SVDB Directory CCI NOPINLOC	Yes
LVS Push Devices SEPARATE PROPERTIES	No, if LVS actionable properties are present.

Detailed seed promotion reporting ([LVS Show Seed Promotions](#) YES) is disabled if device pushdown is enabled explicitly or by default. The hierarchical circuit extractor issues a warning in the transcript when LVS Show Seed Promotions is overridden in this manner.

### Logical Device Equivalence Pushdown

Logical device equivalence pushdown occurs when the LDE option is used.

Logical device equivalence means that device instances are considered equivalent and may be pushed down when the following are true:

- devices have the same set of named properties
- all corresponding properties have the same value
- device pin counts and pin names are the same
- device types (element names) are the same
- device subtypes (model names) are the same

Classifying devices as logically equivalent removes the restriction that the device instances must be formed from the same Device statement in the rule file in order to be candidates for pushdown. The LDE option pushes devices down more freely than with the YES option alone. Consider the following example rule file excerpt:

```
LVS PUSH DEVICES YES //LDE

LAYER red 1
LAYER green 2
LAYER aux1 11
LAYER aux2 12

CONNECT red
CONNECT green

DEVICE R red red(pos) green(neg) <aux1> [ // $D=0
    PROPERTY as,ap,an
    as=area(red)
    ap=area(pos)
    an=area(neg)
]

DEVICE R red red(pos) green(neg) <aux2> [ // $D=1
    PROPERTY as,ap,an
    as=area(red)
    ap=area(pos)
    an=area(neg)
]
```

With LDE not selected, a netlist like the following is produced. For this example, device seeds originated in cell A but were promoted to cell B. Device pushdown did not occur because (otherwise logically equivalent) devices were formed in the TOP cell by different Device statements in the rule file:

```
* SPICE NETLIST
```

```
*****
.SUBCKT A
** N=3 EP=0 IP=0 FDC=0
*.SEEDPROM           $$ devices promoted out of this cell
.ENDS
*****
.SUBCKT B
** N=8 EP=0 IP=6 FDC=4
R0 3 1 as=2.4e-11 ap=2.4e-11 an=2.7e-11 $X=-24000 $Y=-6000 $D=1
R1 4 5 as=2.4e-11 ap=2.4e-11 an=1.2e-11 $X=-24000 $Y=4000 $D=0
R2 6 2 as=2.4e-11 ap=2.4e-11 an=2.7e-11 $X=-7000 $Y=-6000 $D=0
R3 7 8 as=2.4e-11 ap=2.4e-11 an=1.2e-11 $X=-7000 $Y=4000 $D=1
.ENDS
*****
```

Again, the devices in B are formed from different Device statements, as indicated by the \$D=0 and \$D=1 notations on the devices in cell B.

With the LDE option enabled, a netlist like the following is produced:

```
* SPICE NETLIST
*****
.SUBCKT A 4
** N=4 EP=1 IP=0 FDC=2
R0 1 4 as=2.4e-11 ap=2.4e-11 an=2.7e-11 $X=-2000 $Y=-8000 $D=1
R1 2 3 as=2.4e-11 ap=2.4e-11 an=1.2e-11 $X=-2000 $Y=2000 $D=0
.ENDS
*****
.SUBCKT B
** N=2 EP=0 IP=2 FDC=4
X0 1 A $T=-22000 2000 0 0 $X=-24000 $Y=-6000
X1 2 A $T=-5000 2000 0 0 $X=-7000 $Y=-6000
.ENDS
*****
```

Devices are now pushed from cell B into cell A even though they were formed in cell B by different Device statements.

---

#### **Note**



The \$D=<N> notation is arbitrarily chosen on logically equivalent devices that are pushed (LDE is enabled).

---

## SEPARATE PROPERTIES Usage

The **SEPARATE PROPERTIES** functionality is for cases where you need to extract regular LVS properties for comparison and some other properties for device characterization. When these *characterization* properties vary frequently per instance, they are essentially unique per device, so you end up with a flat netlist because that is the only way to represent properties that are different in every instance of a device. However, for LVS-H comparison, your *comparison* properties do not vary from instance to instance, so you do not want a flat netlist in that case.

If you use pushdown backannotation, you create a hierarchical netlist and a flat property netlist using the LVS Push Devices **SEPARATE PROPERTIES** statement and the Calibre

Connectivity Interface (CCI) as discussed under “[Customized Files for Pushdown Backannotation \(PDBA\)](#)” in the *Calibre Query Server Manual*.

LVS Push Devices **YES** must be enabled (true by default), for the LVS Push Devices **SEPARATE PROPERTIES** statement to be accepted. Calibre xRC automatically uses the property netlist generated by the latter statement.

The LVS Push Devices **SEPARATE PROPERTIES** statement directs device pushdown to ignore certain properties and their values when evaluating candidate devices for equivalence during pushdown. The result is that only *LVS-actionable* properties prevent pushdown.

LVS-actionable properties are those required by the LVS comparison phase because they are mentioned in one or more of the following rule file statements:

- [Trace Property](#), including user-defined properties.
- LVS Filter family of statements.
- LVS Reduce family of statements using the TOLERANCE keyword. Only the tolerance properties are used.

Note that properties mentioned in the EFFECTIVE section of LVS Reduce statements *are not* LVS-actionable. This is in keeping with treatment of properties mentioned for these statements themselves.

- [LVS Split Gate Ratio](#)
- [LVS Recognize Gates Tolerance](#)

All other properties are ignored and do not prevent pushdown. The ignored properties are removed from the extracted SPICE netlist and are written to the pushdown properties netlist (generated by CCI) instead.

Suppose the rule file contains statements like these (note that this is not a completely functional rule file; certain items are omitted for clarity):

```
LVS PUSH DEVICES YES

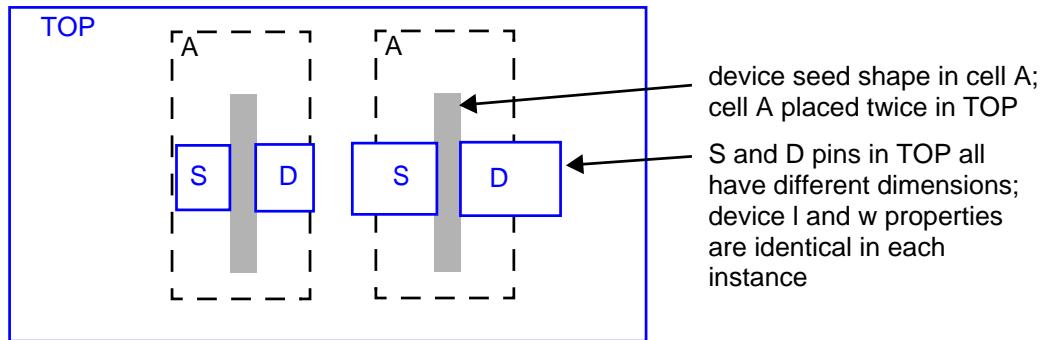
LAYER seed    1
LAYER src_drn 2

CONNECT seed
CONNECT src_drn

DEVICE MN seed seed(G) src_drn(S) src_drn(D) [
  PROPERTY l, w, as, ad
  l = ... // regular length calculation
  w = ... // regular width   calculation
  as = AREA(s)
  ad = AREA(d)
]

TRACE PROPERTY l l 0
TRACE PROPERTY w w 0
```

Assume that there is a single cell A, placed twice in cell TOP. A seed shape originates in A, but all S and D pin shapes are drawn in TOP. The S and D pin sizes in TOP are all different, as shown in the following figure:



The seeds from both placements of A are tentatively promoted to TOP and devices are recognized there. With LVS Push Devices **SEPARATE PROPERTIES** is not used, the devices would remain there because all “as” and “ad” property values would differ.

When this statement appears in the rules:

#### LVS Push Devices SEPARATE PROPERTIES

the only properties that are relevant when determining device equivalence during pushdown are l and w. Because the l and w values are the same across tentatively-promoted devices in this case, they do not prevent pushdown. Properties “as” and “ad” are not LVS-actionable; they no longer prevent pushdown because of the added specification statement. Instead, the “as” and “ad” values are pruned from the extracted netlist and written to the pushdown properties netlist in the SVDB. This file is recognized automatically by Calibre xRC.

Here is an example file produced by LVS Push Devices **SEPARATE PROPERTIES** in conjunction with the LAYOUT NETLIST SEPARATED PROPERTIES YES and LAYOUT SEPARATE PROPERTIES WRITE commands in CCI:

```
* Calibre pushdown separate-properties (back-annotation) data
*****
*
* These lvs-actionable properties, indexed by DEVICE element(model) will
be kept in the netlist:
*
* KEEP_PROP_BEGIN
* MN l w
* KEEP_PROP_END
*
.SUBCKT A
* nothing transcribed
.ENDS
*****
.SUBCKT TOP
X0/M0 as=2e-16 ad=3e-16 $D=0
X1/M0 as=3e-16 ad=4e-16 $D=0
.ENDS
*****
```

The upper portion of the file includes a KEEP\_PROP\_BEGIN ... KEEP\_PROP\_END section. This lists all LVS-actionable properties by device type and subtype. These properties will be kept in the extracted layout netlist. All other properties are removed from the extracted netlist and written to the pushdown properties netlist.

The file also shows the tentatively-promoted sites where devices were recognized and the full hierarchical path to the final device location after device pushdown (X0/M0 and X1/M0 in this case). Because the removed properties may have had different values at tentatively-promoted recognition sites, such properties are included with each hierarchical path shown in the file.

Here is an example extracted netlist:

```
* SPICE NETLIST
*****
*
.SUBCKT A 1 2 3
M0 3 1 2 N l=5e-08 w=5e-08 $D=0
.ENDS
*****
.SUBCKT TOP
X0 1 2 3 A $X=-1 $Y=0
X1 4 5 6 A $x=22 $Y=0
.ENDS
*****
```

Notice the hierarchy of the layout is preserved and this netlist can be used for hierarchical LVS comparison. The as and ad properties do not appear here because they have been ignored for pushdown and are written in the pushdown properties netlist shown previously.

By default, MOS source-drain properties AS, AD, PS, and PD are swappable. Other MOS source-drain characterization properties are not considered swappable by default. To cause other MOS source-drain properties to become swappable, use the [LVS MOS Swappable Properties](#) statement.

When SEPARATE PROPERTIES “*filename*” is specified and pin location information is requested ([Mask SVDB Directory](#) with the CCI option), pin locations are written to *filename* in a format similar to the following. Note the \$PIN\_XY field:

```
X0/M1 AS=1 AD=2 PS=3 PS=4 $PIN_XY=10,10,20,20,30,30 $D=0
```

For built-in MOS devices, Calibre swaps PIN\_XY values for source and drain pins when devices have their source/drain pins swapped during device pushdown. This is reflected in the file that is generated. (Recall that Calibre also swaps known built-in property values such as AS/AD, PS/PD, if present, when the related source/drain pins are swapped.)

Calibre can detect generic user-defined MOS devices and perform the same swapping of PIN\_XY values, and any AS/AD, PS/PD property values when source/drain pins are swapped during pushdown.

A generic user-defined MOS device can only be formed during Calibre device recognition from a [Device](#) statement that satisfies all of the follow conditions:

- The Device statement must contain one each of the following pins, in any order: G, S, D.
- The S and D pins must be swappable.
- There must be only one swap group in the Device statement; that is, no other pins besides S/D can be swappable with any other pin.
- There can be any number of additional pins specified in the Device statement.
- There can be any number of auxiliary layers specified.
- The Device statement can use any user-defined element name and any model name (model name is optional).

For generic user-defined MOS device, built-in swappable properties (AS/AD, PS/PD and so on — the same set supported for built-in MOS) are supported, when present.

When S/D pins are swapped during pushdown, these property values are swapped just as with built-in MOS devices.

The following DEVICE statement will be interpreted as a generic user-defined MOS device. The appropriate swapping is performed during device pushdown:

```
DEVICE userMOS seedLayer seedLayer(G) layerSD(S) layerSD(D) layB(B)
```

Calibre also detects fully generic swappable user devices and swaps PIN\_XY values when related pins are swapped during device pushdown. No property value swapping is supported. Fully generic swappable user devices can only be formed during Calibre device recognition from a Device statement that satisfies all of the following conditions:

- The Device statement must contain exactly two pins of any name that are swappable with one another.
- There must be only one swap group in the Device statement; that is, no other pins besides the pair satisfying the first requirement can be swappable with any other pin.
- There can be any number of additional pins specified in the Device statement.
- There can be any number of auxiliary layers specified.
- The Device statement can use any user-defined element name and any model name (model name is optional).

The following Device statement will be interpreted as a fully generic swappable user device. The appropriate swapping is performed during device pushdown:

```
DEVICE userR seedLayer pinLayer(POS) pinLayer(NEG)
```

**Note**

 **LVS Property Initialize** does not apply in the context of LVS Push Devices **SEPARATE PROPERTIES** because LVS Property Initialize does not make initialized properties LVS-actionable on their own.

---

## Generating Pin Location Information

You generate pin location information by using the LVS Push Devices **SEPARATE PROPERTIES** statement in conjunction with the [Mask SVDB Directory](#) statement containing any of these keywords: CCI, PINLOC, or XRC.

For example, if you specify all of these statements in the rule file:

```
LVS PUSH DEVICES YES // enabled by default
LVS PUSH DEVICES SEPARATE PROPERTIES
MASK SVDB DIRECTORY "svdb" QUERY CCI PINLOC
```

then pin locations are written to the *svdb* directory.

The syntax of pin locations is similar to that produced by the Query Server LAYOUT NETLIST PIN LOCATIONS YES and LAYOUT NETLIST WRITE commands. For additional information, see “[Commands for Generating Customized SPICE Netlists](#)” in the *Calibre Query Server Manual*.

The syntax is shown in the following example. Notice the strings that begin with “\$PIN\_XY=”. The numbers that follow this string are the x,y coordinate pairs of the lower-left corners of the pin shapes in the order the pins are specified in the rule file Device statement. The corresponding Device statement is indicated by \$D=n, where n is a non-negative integer indicating the order of the Device statements in the rule file. For example, \$D=0 means the first Device statement in the rule file was used to define the element shown in the netlist.

```
* Calibre pushdown separate-properties (back-annotation) data
*****
*
* These lvs-actionable properties, indexed by DEVICE element(model) will
be kept in the netlist:
*
* KEEP_PROP_BEGIN
* KEEP_PROP_END
*
.SUBCKT A
* nothing transcribed
.ENDS
*****
.SUBCKT Z
X0/M0 l=2.66667e-08 w=3e-08 as=1.187e-15 ad=6e-16 ps=1.5e-07 pd=1e-07
str="na" $PIN_XY=0,20,0,20,20,20 $D=0
X1/M0 l=2.66667e-08 w=3e-08 as=6e-16 ad=4e-17 ps=1e-07 pd=8.2e-08 str="na"
$PIN_XY=175,-110,175,-95,195,-95 $D=0
.ENDS
*****
.SUBCKT D
* nothing transcribed
.ENDS
```

```
*****
.SUBCKT C
* nothing transcribed
.ENDS
*****
.SUBCKT B
X0/M0 l=2.75862e-08 w=2.9e-08 as=6e-16 ad=1.14e-16 ps=1e-07 pd=1.18381e-07
str="na" $PIN_XY=307,102,327,112,326,92 $D=0
X3/X0/M0 l=3.07692e-08 w=2.6e-08 as=7.88e-16 ad=6e-16 ps=1.74e-07 pd=1e-07
str="na" $PIN_XY=0,16,0,16,20,20 $D=0
X3/X1/X1/M0 l=2.96296e-08 w=2.7e-08 as=9.05e-17 ad=6e-16 ps=1.0316e-07
pd=1e-07 str="na" $PIN_XY=35,-102,41,-112,55,-92 $D=0
.ENDS
*****
.SUBCKT TOP
X0/M0 l=2.66667e-08 w=3e-08 as=2e-16 ad=6e-16 ps=9.2e-08 pd=1e-07 str="na"
$PIN_XY=0,20,0,20,20,20 $D=0
X1/M0 l=2.66667e-08 w=3e-08 as=6e-16 ad=8e-16 ps=1e-07 pd=1.2e-07 str="na"
$PIN_XY=103,-1,103,14,123,14 $D=0
X2/M0 l=2.66667e-08 w=3e-08 as=6e-16 ad=4e-17 ps=1e-07 pd=8.2e-08 str="na"
$PIN_XY=119,-64,139,-54,139,-74 $D=0
X4/X1/M0 l=2.28571e-08 w=3.5e-08 as=4e-17 ad=1.096e-15 ps=8.2e-08
pd=1.36e-07 str="na" $PIN_XY=557,-129,557,-129,577,-134 $D=0
X4/X3/X1/X0/M0 l=8e-08 w=1e-08 as=1.098e-15 ad=6e-16 ps=1.58e-07 pd=1e-07
str="na" $PIN_XY=257,-158,257,-158,277,-146 $D=0
.ENDS
*****
```

## Application of SEPARATE PROPERTIES Option to LVS Property Map

Recall that **LVS Property Map** relates property names specified in LVS Reduce ... TOLERANCE statements to names actually present in the layout and source input databases. Such input database property names may be different in layout and source. Any single property name mentioned in a LVS Reduce ... TOLERANCE statement can be mapped to the (possibly differing) property names present in layout and source using LVS Property Map. Refer to the descriptions of the LVS Property Map *target\_property* and *database\_property* parameters for further details.

For LVS Push Devices **SEPARATE PROPERTIES**, LVS Property Map ... LAYOUT is used to determine the layout *database\_property* names to be kept automatically in the extracted layout netlist based on the *target\_property* names present in LVS Reduce ... TOLERANCE statements, and based upon both the *database\_property* names and *target\_property* names present in relevant LVS Property Map ... LAYOUT statements.

Example:

```
// assume a property in layout is named: WL
// assume a property in source is named: WS

LVS REDUCE ... MN ... [ TOLERANCE W 0 ]

LVS PROPERTY MAP MN W WL LAYOUT
LVS PROPERTY MAP MN W WS SOURCE

LVS PUSH DEVICES SEPARATE PROPERTIES "file"
```

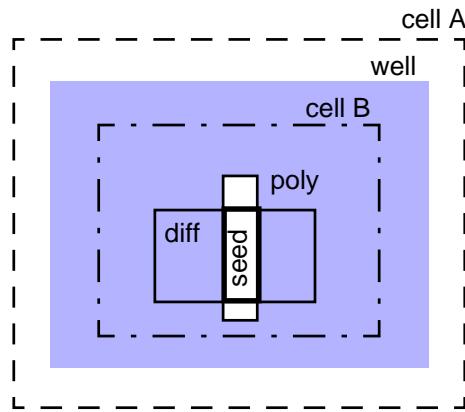
The system observes TOLERANCE W in the LVS Reduce statement, relates this property to WL in the LVS Property Map ... LAYOUT statement, and causes WL to be kept in the extracted layout netlist automatically.

### Examples

#### Example 1

The following example illustrates recovery when pin shapes are present only outside a cell. In [Figure 4-162](#), the seed polygon is in cell B, but the well pin polygon is present only at a higher level of hierarchy, in cell A. The device is initially promoted from cell B to cell A, but later is pushed back into cell B and the copy in A is removed.

**Figure 4-162. Seed Promotion**



Rule file:

```
SD = DIFF NOT POLY
SEED = POLY AND DIFF
CONNECT METAL POLY SD BY CONT

DEVICE MP SEED POLY(G) SD(S) SD(D) WELL(B)
```

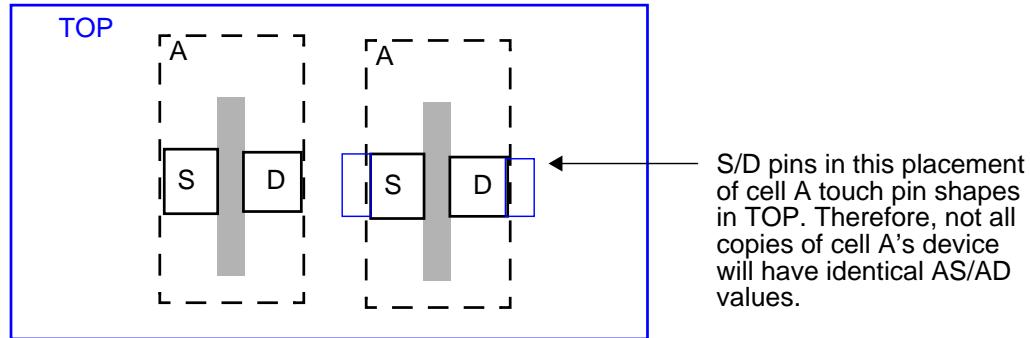
#### Example 2

The following example illustrates recovery from property induced promotion. Consider the following Device operation:

```
DEVICE MP SEED POLY(G) SD(S) SD(D) [
    PROPERTY AS, AD
    AS = AREA(S)
    AD = AREA(D)
]
```

Consider a cell containing a transistor, with the SD polygons in the cell touching SD polygons from outside the cell. The promotion occurs initially and copies of the device are formed, but then an attempt is made to push those copies back down into the cell. However, this pushdown

can occur only if all individual device copies have identical AS/AD values up the hierarchy. Often, that may not be the case, and devices seeds will remain promoted up the hierarchy.



## LVS Recognize Gates

Specification statement

**LVS RECOGNIZE GATES {ALL | SIMPLE | NONE}**  
[MIX SUBTYPES] [XALSO] [WITHIN TOLERANCE]  
[CELL LIST *list\_name*]

### Summary

Specifies whether to recognize logic gates from transistor-level data. Input pins are swappable in logic gate recognition.

### Parameters

- **ALL | SIMPLE | NONE**

A keyword set that indicates the types of gates that the tool recognizes. One of these keywords must be specified. Possible choices are:

**ALL** — Specifies that all gates are recognized.  
**SIMPLE** — Specifies that simple gates are recognized.  
**NONE** — Specifies that no gates are recognized.

- **MIX SUBTYPES**

An optional keyword allows subtype mixing of transistors in series-parallel structures. This optional keyword alters the behavior of the specified ICtrace variable as follows:

**ALL** becomes

```
lvs_recognize_gates = TRUE  
lvs_recognize_only_simple_gates = FALSE
```

**SIMPLE** becomes

```
lvs_recognize_gates = TRUE  
lvs_recognize_only_simple_gates = TRUE
```

**NONE** becomes

```
lvs_recognize_gates = FALSE  
lvs_recognize_only_simple_gates = FALSE
```

- **XALSO**

An optional keyword that allows MOS transistors with X+ subtypes to be included in logic gate recognition. Normally, such devices defined as follows:

```
DEVICE M(XP) gate poly sd sd well      //Rule file  
DEVICE M(XABC) gate poly sd sd well    //Rule file  
  
M1 1 2 3 4 XP                         //Netlist  
M2 1 2 3 4 XABC                        //Netlist
```

are not included in logic gate recognition, but devices such as the following are:

```
DEVICE M(X) gate poly sd sd well      //Rule file
M3 1 2 3 4 X                         //Netlist
```

- **WITHIN TOLERANCE**

An optional secondary keyword that specifies logic gate recognition should use tolerance checking, as specified in [LVS Recognize Gates Tolerance](#) specification statements. The WITHIN TOLERANCE keyword enables tolerance checking when the keywords ALL or SIMPLE are also specified. It has no effect when NONE is specified.

- **CELL LIST *list\_name***

Optional keyword and name of a list from an [LVS Cell List](#) specification statement. This keyword set indicates that the LVS Recognize Gates specification statement in which the cell list appears applies only to cells in the specified list (the cell names are source schematic cell names). If the CELL LIST option is specified, it must be the final parameter of the LVS Recognize Gates specification statement.

## Description

Specifies whether to recognize logic gates from transistor-level data. The default is for ALL gates to be recognized that are known to the LVS system. The input pins are swappable in logic gate recognition. For details about gate recognition concepts, see the section “Logic Gate Recognition” in the [Calibre Verification User’s Manual](#).

This statement may be specified more than once for statements having differing CELL LIST parameters. If this statement has no CELL LIST parameter, then it applies to all gates that are not a part of statements that do have CELL LIST parameters.

[LVS Ground Name](#) and [LVS Power Name](#) must be specified for gates to be recognized. Specifying SIMPLE means gates that are *not* AOI and OAI, as well as higher level serial-parallel structures of type SPxxx(*expression*), are recognized.

Transistors in a series-parallel structure must have the same subtype if MIX SUBTYPES is not specified. Series-parallel structures include the pullup and pulldown sections of logic gates (separately), serial up and serial down structures; series-parallel up and series-parallel down structures; Sm\* and SPm\* structures; and so forth.

Specifying NONE forces the SAME ORDER option for [LVS Reduce Split Gates](#).

[LVS Filter Unused Option](#) INV filters inverter gates that meet the appropriate criteria. Such gates are not recognized.

The CELL LIST option causes gate recognition to occur only within the cells in the *list\_name*. If you use the CELL LIST option, then gate recognition is applied to all source cells in the list and their corresponding layout cells, as follows:

- If there is only one layout cell corresponding to a given source cell, then the cell-specific gate recognition options are applied to both source and layout cells.

- If there are several layout cells corresponding to a given source cell, then the cell-specific gate recognition options are applied to the source cell and all corresponding layout cells.
- If there are several source cells corresponding to one layout cell, and only one of these source cells has cell-specific gate recognition options, then the gate recognition options are applied to the given source cell, corresponding layout cell, and all other source cells corresponding to that layout cell.
- If there are several source cells corresponding to one layout cell, and more than one of these source cells has cell-specific gate recognition options, then one of the source cells with cell-specific options is chosen randomly and its options are applied to the corresponding layout cell and to all corresponding source cells without cell-specific gate recognition options. Note that this condition may be reported as an error in future versions of Calibre.
- If a cell name in the cell list refers to a layout cell, then this name is ignored unless it also matches a source cell, in which case it is used for that source cell as usual. No warning is currently given when a cell name refers to a layout cell but not to a source cell.
- If a cell name in the cell list refers to a cell that is not an hcell, an error is reported as described in the LVS Cell List statement.

When [LVS Inject Logic](#) YES is specified (which it is by default), certain structures are injected into the design. These structures do not participate in logic gate recognition. However, they do participate in logic gate recognition when logic injection is turned off. Therefore, you may notice differences in the LVS report for various injected or logic gate structures, depending upon what is enabled in the rule file. See “[Effects of Logic Injection](#)” in the *Calibre Verification User’s Manual* for more information.

You can specify gate recognition behavior from the LVS Options pane in Calibre Interactive—nmLVS. See “[Controlling Logic Gate Recognition](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In Pyxis Layout, this statement sets the values of application variables `lvs_recognize_gates` and `lvs_recognize_only_simple_gates`, depending on the optional MIX SUBTYPES keyword. The application variable `lvs_recognize_gates` specifies whether to prevent the formation of complex gates. Complex gates include AOI and OAI gates as well as higher level serial-parallel structures of type `SPxxx(expression)`, where `xxx` is a string of characters. The value of this variable is used only when `lvs_recognize_gates` is true.

## Examples

### Example 1

```
LVS POWER NAME VDD
LVS GROUND NAME VSS

// Recognize all logic gates.
LVS RECOGNIZE GATES ALL
```

**Example 2**

```
LVS POWER NAME VDD
LVS GROUND NAME VSS

// Recognize all logic gates for cells not in cell list analog.
LVS RECOGNIZE GATES ALL

// Do not recognize gates for this list
LVS CELL LIST analog analog_block
LVS RECOGNIZE GATES NONE CELL LIST analog
```

# LVS Recognize Gates Tolerance

Specification statement

```
LVS RECOGNIZE GATES TOLERANCE component_type [ ( component_subtype ) ]  
    property_name [ ( spice_parameter ) ]  
    { { SERIES tolerance | PARALLEL tolerance |  
        SERIES tolerance PARALLEL tolerance |  
        tolerance } |  
        { STRING [SERIES] [PARALLEL] } }  
    [SOURCE] [LAYOUT]
```

Used only in Calibre nmLVS/nmLVS-H and PERC.

## Summary

Specifies to limit logic gate recognition based upon property matching using numeric value tolerances or string comparison. When tolerance checking in logic gate recognition is enabled, Calibre nmLVS checks property values of transistors to determine if they are logically equivalent and potentially swappable. If so, and if the transistor properties can be matched, then gate recognition occurs. Tolerance checking is enabled using the [LVS Recognize Gates WITHIN TOLERANCE](#) keyword.

## Parameters

- *component\_type*

A required [Device](#) component type for MOS transistors. Expected component types are M, MD, ME, MN, MP, LDD, LDDD, LDDE, LDDN, LDDP, or an equivalent device specified in an [LVS Device Type](#) statement.

Component types other than these may be specified, but such devices do not participate in gate recognition; LVS simply attempts to read the property values from non-MOS devices. If it can, the components and properties are silently ignored by gate recognition. If not, property errors are written to the LVS report.

- ( *component\_subtype* )

An optional LVS component subtype, which must be enclosed in parentheses, to which this statement applies. When the subtype is specified, the statement only applies to device instances that match the *component\_type* and *component\_subtype*. When the *component\_subtype* parameter is not specified, then the statement applies to all instances of the *component\_type* that do not have a *component\_subtype* specified in any LVS Recognize Gates Tolerance statement.

- *property\_name*

A required property name of a property to be checked.

- ( *spice\_parameter* )

An optional string in parentheses that is an upper or lowercase letter. It is specified with the *property\_name* parameter. It may not be specified with the **STRING** keyword. The letter

instructs LVS to treat property values as strings in SPICE-like syntax (see section “SPICE-Like Property Syntax” in the *ICverify User’s and Reference Manual*) and to parse the strings to obtain the specified SPICE value. Possible parameters are as follows:

<b>W</b> MOS width	<b>C</b> capacitance
<b>L</b> MOS length	<b>A</b> diode area
<b>R</b> resistance	<b>P</b> diode perimter

See the **component\_type** parameter description of how non-MOS devices are handled.

When this parameter is present, the statement applies only to instances that have a property with the specified **property\_name** parameter. The property value contains the specified **spice\_parameter**.

- **SERIES tolerance | PARALLEL tolerance | SERIES tolerance PARALLEL tolerance | tolerance**

A set of keywords and parameters that specify the property to be checked is numeric. Either a selection from this set or the **STRING** keyword must be specified, but not both.

**tolerance** — a non-negative floating-point numeric that represents a percentage. For example, a tolerance of 10 means the numeric value corresponding to the **property\_name** is checked to within a 10% tolerance. When specified without the **SERIES** or **PARALLEL** keywords, the tolerance applies to both **SERIES** and **PARALLEL** cases.

**SERIES** — keyword that specifies the associated **tolerance** applies to transistors connected in series. May be specified with **PARALLEL**.

**PARALLEL** — keyword that specifies the associated **tolerance** applies to transistors connected in parallel. May be specified with **SERIES**.

- **STRING [SERIES] [PARALLEL]**

A keyword that specifies the property to be checked is a string. If **STRING** is specified alone, it applies to both **SERIES** and **PARALLEL** cases. Either **STRING** or a keyword from the **tolerance** set must be specified, but not both. **STRING** may not be specified with (**spice\_parameter**).

**SERIES** — an optional keyword specified with **STRING** that causes the property checking to apply to series transistors. May be specified with **PARALLEL**.

**PARALLEL** — an optional keyword specified with **STRING** that causes the property checking to apply to parallel transistors. May be specified with **SERIES**.

- **SOURCE**

An optional keyword that specifies the statement applies to the source design. This option is used by default.

- **LAYOUT**

An optional keyword that specifies the statement applies to the layout design. This option is used by default.

### Description

This statement specifies that Calibre nmLVS limits formation of logic gates to transistors that have numeric property values within the specified ***tolerance***, or that have matching string values. When enabled, this statement applies to all logic gates recognized by LVS. See the section “[Logic Gate Recognition](#)” in the *Calibre Verification User’s Manual* for more information.

For each combination of ***component\_type***, ***component\_subtype***, ***property\_name***, and LAYOUT or SOURCE keywords, there can be at most one LVS Recognize Gates Tolerance statement.

The ***component\_type*** is case-sensitive if the [LVS Compare Case](#) specification statement has been specified with the YES or TYPES keyword. Similarly, the ***component\_subtype*** is case-sensitive if the YES or SUBTYPES keyword is specified.

Property tolerance checking is disabled by default, even if the an LVS Recognize Gates Tolerance statement is present in the rule file. To enable tolerance checking, you must also specify [LVS Recognize Gates](#) with the WITHIN TOLERANCE keyword.

When tolerance checking is enabled, Calibre nmLVS checks property values of transistors to determine if they are logically equivalent and potentially swappable. If so, then tolerance checking proceeds according to how the tolerance checking is specified in the statements in the rule file.

If a potential gate structure contains devices of different subtypes, then only the devices of the same subtype are compared with each other. Tolerance requirements between devices of different subtypes are assumed to be met.

For numeric tolerance checking, that is, the ***tolerance*** parameter is specified, the transistors of the specified ***component\_type*** and optional ***component\_subtype*** are only recognized as part of the same gate if they own the ***property\_name***, and the property has numeric values that differ within the ***tolerance*** percentage number. The formula for calculating the percentage of difference between property values is  $\text{abs}((v1-v2)/v1)*100$ , where  $\text{abs}()$  is the absolute value function and v1 and v2 are the property values being compared. When several properties are specified for the given ***component\_type*** and ***component\_subtype***, each property is checked separately.

Within a potential gate structure, the **SERIES** ***tolerance*** keyword is used to compare transistors connected in series, and the **PARALLEL** ***tolerance*** keyword is used to compare transistors connected in parallel. If only one of these keywords is specified, then only the corresponding part of the gate is checked for property differences. If only the ***tolerance*** parameter is specified, then it applies to both series and parallel structures.

For example:

```
LVS RECOGNIZE GATES TOLERANCE MP W SERIES 0 PARALLEL 0
LVS RECOGNIZE GATES ALL WITHIN TOLERANCE CELL LIST b
LVS CELL LIST b blk_2
LVS RECOGNIZE GATES ALL CELL LIST a
LVS CELL LIST a blk_1
```

The first statement defines a tolerance of 0% for the W property of MP devices in series and parallel structures. Logic gates are not formed if MP transistors are of different widths for cells in list “b.” For cells in list “a,” logic gates are formed even if MP transistors have different widths.

For string tolerance checking, that is, the STRING keyword is specified, the transistors are only recognized as part of the same gate if they own the *property\_name* with identical string values. Again, when several properties are specified for the given *component\_type* and *component\_subtype*, each property is checked separately. Case sensitivity of string property comparison is controlled by the [LVS Compare Case](#) specification statement.

The SERIES and PARALLEL keywords may also be specified with STRING. The behavior of these keywords in this context is similar to how numeric tolerances behave. If neither SERIES nor PARALLEL are specified, then STRING tolerance checking is applied to both series and parallel structures.

LVS Recognize Gates Tolerance rules are checked independently in the source and layout, and different rules can be specified for the source and layout using the SOURCE and LAYOUT keywords. By default, the statement applies to both subdesigns.

Tolerance checking in logic gate recognition enforces a restriction related to [LVS Reduce Split Gates](#). If logic gate tolerance checking is enabled, and a split gate structure violates the tolerance in any pertinent LVS Recognize Gates Tolerance statement, then the LVS Reduce Split Gates SAME ORDER option is enforced on the entire split gate structure, even when that option is not explicitly specified.

Logic injection is also affected by the tolerance checking in logic gate recognition. Specifically, when injecting logic gate circuits (see “[LVS Inject Logic](#)”), if logic gate tolerance checking is enabled and an injection structure violates the tolerance in any pertinent LVS Recognize Gates Tolerance statement, then the input pins of the injected circuit are not swappable. Note that circuits with topological swappability, such as SRAM bit cells or parallel device groups, are not affected by this tolerance check since their pins are always swappable.

## Examples

For all of these examples, assume that an LVS Recognize Gates WITHIN TOLERANCE statement is specified.

### Example 1

Allow up to 5% difference in length for MN devices in series structures of logic gates; parallel structures are not checked:

```
LVS RECOGNIZE GATES TOLERANCE MN L SERIES 5
```

### Example 2

The widths must be equal for MN devices in series and parallel structures of logic gates:

```
LVS RECOGNIZE GATES TOLERANCE MN W SERIES 0 PARALLEL 0
```

This is equivalent to:

```
LVS RECOGNIZE GATES TOLERANCE MN W 0
```

### Example 3

For all MP devices other than subtype A, check property L in the parallel portion of every gate for equality. For MP(A) devices, check property W in the series portion of the gate for equality:

```
LVS RECOGNIZE GATES TOLERANCE MP L PARALLEL 0  
LVS RECOGNIZE GATES TOLERANCE MP(A) W SERIES 0
```

If STRING is specified, then the property is assumed to contain string values. Logic gate recognition checks for string equality in series or parallel device groups when constructing gates.

### Example 4

MN devices in series structures of logic gates must have the same string for property P. Parallel structures are not checked for this property.

```
LVS RECOGNIZE GATES TOLERANCE MN P STRING SERIES
```

### Example 5

MP devices in series and parallel structures of logic gates must have the same string for property Q:

```
LVS RECOGNIZE GATES TOLERANCE MP Q STRING SERIES PARALLEL
```

or, simpler:

```
LVS RECOGNIZE GATES TOLERANCE MP Q STRING
```

# LVS Reduce

Specification statement

```
LVS REDUCE component_type [‘(‘component_subtype‘)’]
  {PARALLEL | SERIES pin_name_1 pin_name_2}
  [[YES [CELL LIST list_name]
  ‘[‘
  [TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]
  [TOLERANCE STRING {string_property_name} [{string_property_name}...]]
  [effective_property_computation]
  ‘]’] | NO]
```

## Summary

Provides generic device reduction instructions to LVS.

## Parameters

- *component\_type*

A required [Device](#) component type to which this statement applies. It can be any component type, including user-defined or built-in types. The built-in device types for circuit comparison are discussed in detail in the “Built-In Device Types” section of the [Calibre Verification User’s Manual](#).

When *component\_type* is a built-in device type known to LVS and no *effective\_property\_computation* is present, default effective property calculations are performed for built-in properties. Refer to the examples provided for each of the LVS Reduce Parallel/Series component types (commands [LVS Reduce Parallel Bipolar](#) through [LVS Reduce Split Gates](#)) for information on the default effective property calculations. If you specify an *effective\_property\_computation*, you assume responsibility for calculation of all effective properties for the reduced devices.

If *component\_type* is not a built-in device type and no *effective\_property\_computation* is present, then no effective properties are calculated for reduced devices. Any properties present are set to the “unknown” value. This causes LVS discrepancies if those properties are traced.

- *(component\_subtype)*

An optional LVS component subtype, which must be enclosed in parentheses, to which this statement applies. The LVS Reduce statement applies to all instances of the *component\_type* when you do not specify this parameter, except for subtypes that are involved in specific LVS Reduce statements that apply to built-in devices known to LVS.

- **PARALLEL**

Keyword that instructs LVS to reduce the devices in parallel. For the reduction to occur, the following conditions must apply:

- The devices must have the same optional component subtype, the same number of pins, and the same pin names.
- All pins must be connected to the same nets, respectively.
- Pins may be swapped if they are specified as logically equivalent, which can enable reduction in certain configurations.

- **SERIES *pin\_name\_1* *pin\_name\_2***

Keyword set that instructs LVS to reduce the devices in series. For the reduction to occur, the following conditions must apply:

- The devices must have the same optional component subtype, the same number of pins, and the same pin names.
- The named pins (*pin\_name\_1* and *pin\_name\_2*) must be connected in series in alternate order, for example *pin\_name\_1* to *pin\_name\_2* to *pin\_name\_1*, and so forth.
- All other pins, if present, must be connected in parallel (connected to the same nets, respectively).
- The *pin\_name\_1* and *pin\_name\_2* may be swappable amongst themselves, but may not be swappable with any other pins.

In cases where pin swapping is not allowed, series reduction does not occur. Consider this example:

```
.subckt test
X1 1 2 dev
X2 1 3 dev
.ends
```

This cannot be reduced to X1 2 3 dev unless the pins are swappable.

- **YES | NO**

An optional keyword that instructs LVS whether or not to reduce the specified devices. YES is the default behavior and triggers device reduction. If NO is specified, then reduction does not occur for the specified devices. The YES, CELL LIST, and *effective\_property\_computation* parameters may not be specified with NO.

- **CELL LIST *list\_name***

Optional keyword and name of a list from an [LVS Cell List](#) specification statement. This keyword set indicates that the LVS Reduce specification statement in which the cell list appears applies only to cells in the specified list. The CELL LIST keyword set must follow either the **PARALLEL** or **SERIES** keyword sets, and the YES keyword if specified. The

CELL LIST keyword set must precede the *effective\_property\_computation* if both are specified. See [Cell-Specific Reduction](#).

- [ ]  
You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*. The single set of brackets must surround the entire set of TOLERANCE keyword groups and the *effective\_property\_computation*, if present.
- TOLERANCE *property\_name tolerance\_number*  
An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area), R (resistance), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(w).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction”, in the [Calibre Verification User’s Manual](#).

- TOLERANCE STRING *string\_property\_name*  
An optional keyword set that allows you to limit the reduction of devices that have different string property values. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce statement.  
Devices are not reduced if their *string\_property\_name* parameters do not match. When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.
- *effective\_property\_computation*  
An optional user-created program that defines the calculations to take place during device reduction. If specified, it must immediately follow the YES keyword. It may not be specified with the NO keyword.  
The presence of an effective property computation overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties available by default.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

The following is an example of an effective property computation within an LVS Reduce statement:

```
LVS REDUCE...
[
    effective r, l, w

    p = sum (w*l)
    q = sum (w/l)
    w = sqrt ( p*q )
    l = sqrt ( p/q )

    checkForZero = prod ( r )
    if ( checkForZero == 0 )
        r = 0
    else
        {
            // demonstrate use of braces
            // assign 1/sum(1/r) in two steps:
            recipSum = sum(1/r)
            r = 1 / recipSum
        }
]
```

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

## Description

Specifies whether to perform generic reduction of devices. You specify the parameters of a device within the [Device](#) statement or an [LVS Device Type](#) statement. The types of devices can be user-defined or built-in. LVS Reduce is often referred to as the *generic LVS Reduce statement* because it is not device-specific as other LVS Reduce statements in this family. These statements appear in the LVS Setup section of the LVS report.

The default behaviors of non-generic LVS Reduce specification statements, such as [LVS Reduce Parallel MOS YES](#), *remain unchanged* if you specify a generic LVS Reduce statement involving a specific MOS component type. For example, if the only generic LVS Reduce statement in the rule file is this:

```
LVS REDUCE MP PARALLEL NO
```

the default behavior of the LVS Reduce Parallel MOS specification statement still applies to all MOS devices other than type MP. Similarly, if the only generic LVS Reduce statement in the rule file is:

```
LVS REDUCE MP PARALLEL YES
```

then all MOS devices, including MP devices, are reduced in parallel. The previous statement, in fact, is redundant due to default behavior of LVS Reduce Parallel MOS.

For a complete discussion of device reduction, see “Device Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Builtin MOS NRD\\_NRS](#) for information about NRS and NRD property handling.

## Statement Precedence

Before performing generic LVS Reduce statements, LVS internally organizes them into an order of precedence. The order of precedence is unrelated to the order of the statements within the rule file and is determined in the following order, from highest to lowest:

1. Generic LVS Reduce statements with both *component\_type* and *component\_subtype* specified.
2. Generic LVS Reduce statements with *component\_type* specified but not *component\_subtype*.
3. Built-in LVS Reduce statements relevant to a particular component type, such as the LVS Reduce Parallel MOS specification statement for the component type MP.

Consider the following rule file excerpt:

```
LVS REDUCE SERIES RESISTORS YES          // lowest precedence
LVS REDUCE R SERIES NO                  // next precedence
LVS REDUCE R(a) SERIES [                // highest precedence
    EFFECTIVE r
    r = sum ( r )
]
```

Based on the previous statements, devices would fall into the following groups:

- Devices R(a) reduced with the effective property computation:  $r=\text{sum}(r)$ :
- Devices not reduced:  
R — without the subtype of a.
- The LVS Reduce Series Resistors YES statement behavior is overridden by LVS Reduce R Series NO.

## Cell-Specific Reduction

The CELL LIST keyword set indicates that a particular generic LVS Reduce specification statement applies only to cells in the specified list. All cell names in a [LVS Cell List](#) list are source names. Cell-specific device reduction options are applied to all source cells in the list and their corresponding layout cells.

Note that if a cell name in the LVS Cell List refers to a non-hcell, an error is reported.

## Syntax Restrictions

The following are not allowed when specifying generic LVS Reduce statements.

- You cannot specify two generic LVS Reduce statements containing identical *component\_type*, *component\_subtype* (optional), and reduction type (PARALLEL or SERIES). For example, the following rule file excerpt is not allowed:

```
LVS REDUCE C(a) PARALLEL ...
LVS REDUCE C(a) PARALLEL ...
```

- You cannot specify *pin\_name\_1* and *pin\_name\_2* as the same name. For example, the following rule file excerpt is not allowed:

```
LVS REDUCE M(b) SERIES ABC ABC ...
```

- When you specify a built-in device as a *component\_type* known to LVS, the parameters *pin\_name\_1* and *pin\_name\_2* are restricted to built-in pin names known to LVS. See [Device](#).

MOS Transistor family: S (source) and D (drain)

Bipolar Transistor family: C (collector) and E (emitter)

Capacitor, Resistor, Diode: P (pos) and N (neg)

- The CELL LIST keyword set must follow the **PARALLEL** or **SERIES** keywords, and the YES keyword, if specified. The CELL LIST keyword set must precede the *effective\_property\_computation* if both are specified.
- The NO keyword may not be specified with YES, CELL LIST, or *effective\_property\_computation*.

## Consistency Checks

Several consistency checks are included in the SERIES form of the generic LVS Reduce statement. These checks detect situations that are invalid or erroneous.

A generic LVS Reduce statement may have a set of zero or more corresponding rule file Device operations. A Device operation corresponds to a generic LVS Reduce statement if the LVS Reduce statement applies to devices created by that operation. The following consistency checks against rule file Device operations are performed on all generic LVS Reduce statements. These checks are performed always, regardless of the [Layout System](#) or [Source System](#).

- If you specify the SERIES option in a generic LVS Reduce statement, then both *pin\_name\_1* and *pin\_name\_2* must exist in all rule file Device operations corresponding to that statement.
- If you specify the SERIES option in a generic LVS Reduce statement, then pins *pin\_name\_1* and *pin\_name\_2* may be swappable among themselves but may not be swappable with any other pins. This condition is checked with respect to pin swappability as indicated in each rule file Device operation corresponding to the generic LVS Reduce statement. Note that rule file Device operations may indicate pin swappability explicitly (swap-list parameters) or implicitly (when different pins are formed from a single layer).

The rule file compiler reports errors when these conditions are not satisfied.

The LVS circuit comparison module verifies the correctness of pin names in the input layout and source databases with respect to generic LVS Reduce SERIES specification statements. Instances with incorrect pin names are reported as input errors; comparison is not performed and the overall result returns a status of Not Compared. Specifically, every input database

instance to which a generic LVS Reduce SERIES *pin\_name\_1 pin\_name\_2* statement applies, must satisfy the following conditions:

- The instance must own pins with the names *pin\_name\_1* and *pin\_name\_2*.

Instances that violate this condition are reported under the heading Series Pin Names Not Found in the Layout Input Errors or Source Input Errors sections of the LVS report. For each instance, LVS indicates the component type, optional subtype, instance name, and expected pin names (one pin per line).

- The instance may not own more than one pin called *pin\_name\_1* or more than one pin called *pin\_name\_2*.

Instances that violate this condition are reported under the heading Ambiguous Series Pin Names Specified in the Layout Input Errors or Source Input Errors sections of the LVS report. For each instance, LVS indicates the component type, optional component subtype, instance name, and ambiguous pin names.

- Pins *pin\_name\_1* and *pin\_name\_2* of the instance may not be swappable with any other pins (such as, pins not named in the particular LVS Reduce SERIES statement).

Instances that violate this condition are reported under the heading Incorrect Series Pin Swap Nets in the Layout Input Errors or Source Input Errors sections of the LVS report. For each instance, LVS indicates the component type, optional component subtype, instance name and, offending pin names (one pin per line).

See also the non-generic LVS Reduce statements that follow this section.

## Examples

### Example 1

Reduce user-defined series device elements UU:

```
DEV UU useed in(P1) out(P2) ...
LVS REDUCE UU SERIES P1 P2
// Reduce UU devices in series along pins P1, P2.
[
    TOLERANCE Z 2
    // Allow 2% tolerance in property Z.
    EFFECTIVE Z
    // Compute effective Z values as a sum
    Z = SUM(Z)
]
```

### Example 2

The following table shows how you might use the generic LVS Reduce statement to mimic the code used for some built-in LVS Reduce Series and LVS Reduce Parallel statements. See

[“Statement Precedence”](#) on page 999 for a description of the precedence of these statements in a rule file.

Built-in LVS Reduce statement	Equivalent Generic LVS Reduce statement
LVS REDUCE PARALLEL CAPACITORS YES	LVS REDUCE C PARALLEL
LVS REDUCE SERIES CAPACITORS YES	LVS REDUCE C SERIES POS NEG
LVS REDUCE PARALLEL MOS YES	LVS REDUCE M PARALLEL LVS REDUCE MN PARALLEL LVS REDUCE MP PARALLEL LVS REDUCE ME PARALLEL LVS REDUCE MD PARALLEL LVS REDUCE LDD PARALLEL LVS REDUCE LDDN PARALLEL LVS REDUCE LDDP PARALLEL LVS REDUCE LDDE PARALLEL LVS REDUCE LDDD PARALLEL

#### Note



If you provide your own effective property computations for reduced MOS devices, you must ensure that you include effective property computations for split gates also. User-defined effective properties for split gates are not inherited from parallel MOS property computations.

#### Example 3

In this example, parallel reduction is performed for all instances of type C, except for those with subtype a. Note that the order of these statements is not important. As shown, a statement that designates only the component type is overridden by a more specific statement that designates the component type and subtype.

```
LVS REDUCE C(a) PARALLEL NO
// exception or override statement
LVS REDUCE C PARALLEL YES // general statement
```

#### Example 4

In this example, parallel reduction is performed for all instances of type C. For instances of type C and subtype a, reduction computation *effective-property-computation-2* is used to calculate effective property values on reduced instances. For all other C-type instances, reduction computation *effective-property-computation-1* is used for effective property values. Note that the second statement overrides the first for instances with type C and subtype a.

```
LVS REDUCE C PARALLEL [ effective-property-computation-1 ]
LVS REDUCE C(a) PARALLEL [ effective-property-computation-2 ]
```

**Example 5**

In this example, parallel reduction is performed for all capacitors, except those with subtype a. Series reduction is performed for all capacitors without exception.

```
LVS REDUCE C PARALLEL  
LVS REDUCE C SERIES POS NEG  
LVS REDUCE C(a) PARALLEL NO
```

## LVS Reduce Parallel Bipolar

Specification statement

### LVS REDUCE PARALLEL BIPOLAR {YES | NO}

```
'['  
[TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]]  
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]  
[effective_property_computation]  

```

### Parameters

- **YES**

Keyword that instructs LVS to reduce parallel bipolar transistors. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs LVS not to reduce parallel bipolar transistors.

- [ ]

You must include *one* set of square brackets if you specify a TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE *property\_name tolerance\_number***

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Parallel Bipolar statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area), W (width), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(w).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- **TOLERANCE STRING** *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one Tolerance STRING keyword in an LVS Reduce Parallel Bipolar statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- **effective\_property\_computation**

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

## Description

Specifies whether to reduce bipolar transistors connected in parallel into single transistors. The default is YES. This statement can be specified once in your rule file.

LVS reports reduction tolerance settings in the LVS Parameters section of the LVS Report as shown in the following example:

```

o LVS Setup:
...
LVS REDUCE PARALLEL BIPOLAR      YES [ TOLERANCE W 1 ]
...

```

In Pyxis Layout, this statement sets the value of application variable `lvs_reduce_parallel_bipolar`.

For more information, refer to “Parallel Bipolar Transistor Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

## Examples

The following example is equivalent to the default built-in effective property calculation for parallel bipolar transistor devices:

```
LVS REDUCE PARALLEL BIPOLAR YES
// Reduce MOS devices in parallel.
[ effective A, W, L
// Calculate these effective values.
P = sum (W * L) // Sum of Wi * Li
Q = sum (W / L) // Sum of Wi / Li
W = sqrt (P * Q) // Effective width
L = sqrt (P / Q) // Effective length
A = sum ( A )    // Effective area: sum of Ai
]
```

# LVS Reduce Parallel Capacitors

Specification statement

## LVS REDUCE PARALLEL CAPACITORS {YES | NO}

```
'['
[TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]
[effective_property_computation]
']'
```

### Parameters

- **YES**

Keyword that instructs LVS to reduce parallel capacitors. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs LVS not to reduce parallel capacitors.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE *property\_name tolerance\_number***

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Parallel Capacitors statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area), C (capacitance), and P (perimeter). This property can also be a property name/SPICE parameter combination, such as instpar(c).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- TOLERANCE STRING *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Parallel Capacitors statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- *effective\_property\_computation*

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

### Description

Specifies whether to reduce capacitor devices connected in parallel into single capacitors. The default is YES. This statement can be specified once in your rule file.

LVS reports reduction tolerance settings in the LVS Parameters section of the LVS Report, as shown in the following example:

```
o LVS Setup:  
...  
LVS REDUCE PARALLEL CAPACITORS      YES [ TOLERANCE C 1 ]  
...
```

In Pyxis Layout, this statement sets the value of application variable `lvs_reduce_parallel_capacitors`.

For more information, refer to “Parallel Capacitor Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

## Examples

The following example is equivalent to the default built-in effective property calculation for parallel capacitor devices:

```
LVS REDUCE PARALLEL CAPACITORS YES
// Reduce capacitor devices in parallel.
[ effective C, A, P
// Calculate these effective values.
C = sum ( C ) // Effective capacitance: sum of Ci
A = sum ( A ) // Effective area: sum of Ai
P = sum ( P ) // Effective perimeter: sum of Pi
]
```

## LVS Reduce Parallel Diodes

Specification statement

### LVS REDUCE PARALLEL DIODES {YES | NO}

```
'['  
[TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]]  
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]  
[effective_property_computation]  

```

### Parameters

- **YES**

Keyword that instructs LVS to reduce parallel diodes. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs LVS not to reduce parallel diodes.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE *property\_name tolerance\_number***

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Parallel Diodes statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area) and P (perimeter). This property can also be a property name/SPICE parameter combination, such as instpar(p).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- **TOLERANCE STRING *string\_property\_name***

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Parallel Diodes statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- ***effective\_property\_computation***

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

## Description

Specifies whether to reduce diodes connected in parallel into single diodes. The default is YES. This statement can be specified once in your rule file.

LVS reports reduction tolerance settings in the LVS Parameters section of the LVS Report, as shown in the following example:

```

o LVS Setup:
...
LVS REDUCE PARALLEL DIODES      YES [ TOLERANCE P 1 ]
...

```

In Pyxis Layout, this statement sets the value of application variable `lvs_reduce_parallel_diodes`.

For more information, refer to “Parallel Diode Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

## Examples

The following example is equivalent to the default built-in effective property calculation for parallel diode devices:

```
LVS REDUCE PARALLEL DIODES YES
// Reduce diode devices in parallel.
[ effective A, P
// Calculate these effective values.
A = sum ( A ) // Effective area: sum of Ai
P = sum ( P ) // Effective perimeter: sum of Pi
]
```

# LVS Reduce Parallel MOS

Specification statement

## LVS REDUCE PARALLEL MOS {YES | NO}

```
'['
[TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]
[effective_property_computation]
']'
```

### Parameters

- **YES**

Keyword that instructs LVS to reduce parallel MOS transistors. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs LVS not to reduce parallel MOS transistors.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE *property\_name tolerance\_number***

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Parallel MOS statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area), W (width), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(w).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- TOLERANCE STRING *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Parallel MOS statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- *effective\_property\_computation*

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

### Description

Specifies whether to reduce MOS transistors connected in parallel into single transistors. The default is YES. This statement can be specified once in your rule file.

LVS reports reduction tolerance settings in the LVS Parameters section of the LVS Report, as shown in the following example:

```
o LVS Setup:  
...  
LVS REDUCE PARALLEL MOS           YES [ TOLERANCE W 1 ]  
...
```

Since parallel MOS devices are a special case of split gates of length one, split gate reduction implies parallel MOS reduction. That is, LVS Reduce Parallel MOS NO behavior is superceded by the default [LVS Reduce Split Gates](#) YES statement. To be effective, a LVS Reduce Parallel MOS NO statement requires the presence of a corresponding LVS Reduce Split Gates NO statement.

---

#### Note



If you provide your own effective property computations for reduced MOS devices, you must ensure that you also include effective property computations for split gates. User-defined effective properties for split gates are not inherited from parallel MOS property computations.

---

In Pyxis Layout, this statement sets the value of application variable lvs\_reduce\_parallel\_mos. For more information, refer to “Parallel MOS Transistor Reduction” in the *Calibre Verification User’s Manual*.

See also [LVS Reduce](#), [Device](#), [LVS Device Type](#), and [LVS Built-in MOS NRD\\_NRS](#).

## Examples

The following example is equivalent to the default built-in effective property calculation for parallel MOS devices. The [LVS Reduce Split Gates](#) statement must also be included, as properties for split gates are not inherited from the LVS Reduce Parallel MOS statement:

```
LVS REDUCE PARALLEL MOS YES
// Reduce MOS devices in parallel.
[ effective W, L, AS, AD, PS, PD
// Calculate these effective values.
P = sum( W * L ) // Sum of Wi * Li
Q = sum( W / L ) // Sum of Wi / Li
W = sqrt( P * Q ) // Effective W
L = sqrt( P / Q ) // Effective L
AS = sum( AS )    // Effective AS: sum of ASi
AD = sum( AD )    // Effective AD: sum of ADi
PS = sum( PS )    // Effective PS: sum of PSi
PD = sum( PD )    // Effective PD: sum of PDi
]
// Effective property computations must also be included for split gates.
```

## LVS Reduce Parallel Resistors

Specification statement

### LVS REDUCE PARALLEL RESISTORS {YES | NO}

```
'['
[TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]
[effective_property_computation]
']'
```

### Parameters

- **YES**

Keyword that instructs LVS to reduce parallel resistors. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs LVS not to reduce parallel resistors.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE *property\_name tolerance\_number***

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Parallel Resistors statement.

The parameters are:

*property\_name* — A device property to analyze, such as A (area), R (resistance), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(w).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- TOLERANCE STRING *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Parallel Resistors statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- *effective\_property\_computation*

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

## Description

Specifies whether to reduce resistor devices connected in parallel into single resistors. The default is YES. This statement can be specified once in your rule file.

LVS reports reduction tolerance settings in the LVS Parameters section of the LVS Report as shown in the following example:

```

o LVS Setup:
...
    LVS REDUCE PARALLEL RESISTORS           YES [ TOLERANCE R 1 ]
...

```

In Pyxis Layout, this statement sets the value of application variable `lvs_reduce_parallel_resistors`.

For more information, refer to “Parallel Resistor Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

## Examples

The following example is equivalent to the default built-in effective property calculation for parallel resistor devices:

```
LVS REDUCE PARALLEL RESISTORS YES
// Reduce resistor devices in parallel.
[ effective R, W, L
// Calculate these effective values.
R = 1 / sum( 1/R ) // Effective resistance
P = sum( W * L )    // Sum of Wi * Li
Q = sum( W / L )    // Sum of Wi / Li
W = sqrt( P * Q )   // Effective W
L = sqrt( P / Q )   // Effective L
]
```

# LVS Reduce Semi Series MOS

Specification statement

## LVS REDUCE SEMI SERIES MOS {NO | YES}

### Parameters

- **NO**

Keyword that instructs LVS not to reduce semi series MOS transistors. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs LVS to reduce semi series MOS transistors.

### Description

Specifies whether to reduce MOS devices connected in series, including those with bypass nets. The default is NO. You can specify this statement once in your rule file. This statement is independent of the [LVS Reduce Series MOS](#) specification statement.

For more information, refer to “Semi-Series MOS Transistor Reduction” in the [\*Calibre Verification User’s Manual\*](#).

See also [Device](#) and [LVS Device Type](#).

### Examples

#### Example 1

The following example is the default statement:

```
LVS REDUCE SEMI SERIES MOS NO
```

#### Example 2

The following example is equivalent to the default built-in effective property calculation for semi-series MOS transistor devices:

```
LVS REDUCE SEMI SERIES MOS YES
// Reduce MOS transistor devices in series, including bypass nets.
[ effective W, L
// Calculate these effective values.
P = sum( W * L ) // Sum of Wi * Li
Q = sum( L / W ) // Sum of Li / Wi
W = sqrt( P / Q ) // Effective W
L = sqrt( P * Q ) // Effective L
]
```

# LVS Reduce Series Capacitors

Specification statement

## LVS REDUCE SERIES CAPACITORS {YES | NO}

```
'['
[TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]
[effective_property_computation]
']'
```

### Parameters

- **YES**

Keyword that instructs LVS to reduce series capacitors. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs LVS not to reduce series capacitors.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE *property\_name tolerance\_number***

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Series Capacitors statement.

The parameters are:

*property\_name* — A device property to analyze, such as A (area), C (capacitance), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(c).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- TOLERANCE STRING *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Series Capacitors statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- *effective\_property\_computation*

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

## Description

Specifies whether to reduce capacitor devices connected in series into single capacitors. The default is YES. This statement can be specified once in your rule file.

The tool reports reduction tolerance settings in the LVS Parameters section of the LVS report, as shown in the following example:

```

o LVS Setup:
...
    LVS REDUCE SERIES CAPACITORS      YES [ TOLERANCE W 1 ]
...

```

In Pyxis Layout, this statement sets the value of application variable `lvs_reduce_series_capacitors`.

For more information, refer to “Series Capacitor Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

### Examples

The following example is equivalent to the default built-in effective property calculation for series capacitor devices:

```
LVS REDUCE SERIES CAPACITORS YES
// Reduce capacitor devices in series.
[ effective C
// Calculate these effective values.
C = 1 / sum ( 1/C) // Effective capacitance
]
```

# LVS Reduce Series MOS

Specification statement

## LVS REDUCE SERIES MOS {NO | YES}

```
'['
[TOLERANCE {property_name tolerance_number} [{property_name tolerance_number}...]]
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]
[effective_property_computation]
']'
```

### Parameters

- **NO**

Keyword that instructs LVS not to reduce series MOS devices. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs LVS to reduce series MOS devices.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE *property\_name tolerance\_number***

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Series MOS statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area), W (width), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(w).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- TOLERANCE STRING *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Series MOS statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- *effective\_property\_computation*

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

### Description

Specifies whether to reduce MOS devices connected in series. The default is NO. This statement can be specified once in your rule file. This statement is independent of the [LVS Reduce Semi Series MOS](#) specification statement; one or the other, or both, may be performed.

The tool reports reduction tolerance settings in the LVS Parameters section of the LVS Report, as shown in the following example:

```
o LVS Setup:  
...  
    LVS REDUCE SERIES MOS           YES [ TOLERANCE W 1 ]  
...
```

For more information, refer to “Series MOS Transistor Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

### Examples

#### Example 1

The following example is the default statement:

```
LVS REDUCE SERIES MOS NO
```

**Example 2**

The following example is equivalent to the default built-in effective property calculation for series MOS transistor devices:

```
LVS REDUCE SERIES MOS YES
// Reduce MOS transistor devices in series.
[ effective W, L
// Calculate these effective values.
P = sum( W * L ) // Sum of Wi * Li
Q = sum( L / W ) // Sum of Li / Wi
W = sqrt( P / Q ) // Effective W
L = sqrt( P * Q ) // Effective L
]
```

## LVS Reduce Series Resistors

Specification statement

### LVS REDUCE SERIES RESISTORS {YES | NO}

```
'[  
[TOLERANCE {property_name tolerance_number} [{property_name  
tolerance_number}...]]  
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]  
[effective_property_computation]  
']
```

#### Parameters

- **YES**

Keyword that instructs LVS to reduce series resistors. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs the tool not to reduce series resistors.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- **TOLERANCE property\_name tolerance\_number**

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Series Resistors statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area), R (resistance), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(w).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- TOLERANCE STRING *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Series Resistors statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- *effective\_property\_computation*

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file.

For further information about effective property computations, refer to “[Device Reduction Effective Property Computation Language](#)” on page 1851.

## Description

Specifies whether to reduce resistor devices connected in series into single resistors. The default is YES. This statement can be specified once in your rule file.

The tool reports reduction tolerance settings in the LVS Parameters section of the LVS Report, as shown in the following example:

```

o LVS Setup:
...
LVS REDUCE SERIES RESISTORS           YES [ TOLERANCE W 1 ]
...

```

In Pyxis Layout, this statement sets the value of application variable `lvs_reduce_series_resistors`.

For more information, refer to “Series Resistor Reduction” in the [Calibre Verification User’s Manual](#).

See also [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

## Examples

The following example is equivalent to the default built-in effective property calculation for series resistor devices:

```
LVS REDUCE SERIES RESISTORS YES
// Reduce resistor devices in series.
[ effective R, W, L
// Calculate these effective values.
R = sum( R ) // Effective resistance
P = sum( W * L ) // Sum of Wi * Li
Q = sum( L / W ) // Sum of Li / Wi
W = sqrt( P / Q ) // Effective W
L = sqrt( P * Q ) // Effective L
]
```

# LVS Reduce Split Gates

Specification statement

```
LVS REDUCE SPLIT GATES {YES [SEMI ALSO] [SAME ORDER]}
[WITHIN TOLERANCE] [MIX TYPES] [SP ALSO]
[''
[TOLERANCE {property_name tolerance_number} [{property_name
tolerance_number}...]]
[TOLERANCE STRING {string_property_name} [{string_property_name} ...]]
[effective_property_computation]
'']
| NO}
```

## Parameters

- **YES**

Keyword that instructs LVS to reduce split gates. This is the default behavior if you do not include this statement in the rule file. For detailed information about split-gate reduction, refer to “Split Gate Reduction” in the *Calibre Verification User’s Manual*.

- **NO**

Keyword that instructs LVS not to reduce split gates.

- **SEMI ALSO**

An optional keyword that instructs LVS to reduce semi-split gate structures in addition to reducing fully split gates.

- **SAME ORDER**

An optional keyword that instructs LVS to reduce split gates only when the input order is the same in all transistor strings that form the split gate. This option is forced if you specify **LVS Recognize Gates NONE**.

- **WITHIN TOLERANCE**

An optional keyword that specifies split-gate reduction should not occur in cases where the **tolerance** value in an **LVS Split Gate Ratio** statement would report a discrepancy.

- **MIX TYPES**

An optional keyword that specifies a split gate structure may contain transistors with different component types, different numbers of pins, different pin names, or different pin swappability. However, component types, numbers of pins, pin names, and pin swappability, as well as component subtypes, must still be the same in each group of transistors that are reduced together (that is, in each “row” of the split gate).

The presence of devices with different component types, numbers of pins, pin names, or pin swappability anywhere in a split gate structure forces the **SAME ORDER** option for the entire structure.

- SP ALSO

An optional keyword that specifies series-parallel split gate reduction is performed in addition to regular split-gate reduction. This is described in the section “[Series-Parallel Split Gate Reduction](#)” in the *Calibre Verification User’s Manual*.

- [ ]

You must include *one* set of square brackets if you specify either the TOLERANCE or TOLERANCE STRING keyword set, or an *effective\_property\_computation*.

The single set of brackets must surround the specified information from the entire group of statements.

- TOLERANCE *property\_name tolerance\_number*

An optional keyword set that allows you to limit the reduction of devices that have different numeric property values. You can specify more than one pair of parameters, but you may specify only one TOLERANCE keyword in an LVS Reduce Split Gates statement. The parameters are:

*property\_name* — A device property to analyze, such as A (area), W (width), and L (length). This property can also be a property name/SPICE parameter combination, such as instpar(w).

*tolerance\_number* — A positive, floating-point number that specifies the allowable percentage variance between two devices to be reduced.

Devices are not reduced if they own a *property\_name* with different values and the percent difference exceeds *tolerance\_number*.

This keyword may be specified with a TOLERANCE STRING keyword set. The order in which they are specified is not important.

For further information about numeric or string reduction tolerance, refer to section “Tolerance in Device Reduction,” in the *Calibre Verification User’s Manual*.

- TOLERANCE STRING *string\_property\_name*

An optional keyword set that allows you to limit the reduction of devices that have different string property values. Devices are not reduced if their *string\_property\_name* parameters do not match. You can specify more than one *string\_property\_name* parameter, but you may specify only one TOLERANCE STRING keyword in an LVS Reduce Split Gates statement.

When more than one *string\_property\_name* parameter is specified, devices are not reduced if there is no match with any of the string parameters in the list. The check for a match is done for each member of the list.

- *effective\_property\_computation*

An optional user-created program that defines the calculations to take place during device reduction.

The presence of an effective property computation cancels and overrides any built-in effective property computation for the particular device type. When you provide an

effective property computation, you must define formulas for all properties of interest, including built-in properties.

Some effective property computations can become lengthy and may extend over several lines in the rule file. For further information about effective property computations, refer to section [“Device Reduction Effective Property Computation Language”](#) on page 1851.

## Description

Specifies whether to reduce MOS split gates formed as serial-up or serial-down structures into single gate structures. The default is YES. This statement can be specified once in your rule file.

It is frequently preferable to use [LVS Short Equivalent Nodes](#) SPLIT. When that option is used, LVS Reduce Split Gates NO must be specified.

There are two special circumstances to consider for split-gate reduction:

1. The presence of X+ devices anywhere in a split gate structure forces the SAME ORDER option in split gate reduction for the entire structure, unless LVS Recognize Gates XALSO is specified. X+ devices are MOS transistors whose component subtype begins with the letter X or x followed by at least one other character. The SAME ORDER option prevents reduction of split gate structures where the order of inputs is different in different fingers of the split gate.
  2. The presence of devices with different subtypes anywhere in a split gate structure forces the SAME ORDER option in split gate reduction for the entire structure, unless LVS Recognize Gates MIX SUBTYPES is specified.

Split gate reduction implies parallel MOS reduction, so specifying YES results in parallel MOS reduction. Refer to the [LVS Reduce Parallel MOS](#) specification statement.

## **Note**



If you provide your own effective property computations for reduced MOS devices, you must ensure that you include effective property computations for split gates also. User-defined effective properties for split gates are not inherited from parallel MOS property computations.

The tool reports reduction tolerance settings in the LVS Parameters section of the LVS Report, as shown in the following example:

- o LVS Setup:  
...  
LVS REDUCE SPLIT GATES YES [ TOLERANCE W 1 ]  
...

The SEMI ALSO option is automatically enforced, even when not explicitly specified, in cases where [LVS Recognize Gates](#) WITHIN TOLERANCE is specified, and tolerance checking for gate recognition finds a discrepancy resulting from any [LVS Recognize Gates Tolerance](#) statement.

In Pyxis Layout, this statement sets the value of application variable lvs reduce split gates.

See also [LVS Split Gate Ratio](#), [LVS Reduce](#), [Device](#), and [LVS Device Type](#).

### Examples

The following example is equivalent to the default built-in effective property calculation for split gate devices:

```
LVS REDUCE SPLIT GATES YES
// Reduce split gate devices in parallel.
[ effective W, L
// Calculate these effective values.
P = sum( W * L ) // Sum of Wi * Li
Q = sum( W / L ) // Sum of Wi / Li
L = sqrt( P / Q ) // Effective L
W = sqrt( P * Q ) // Effective W
]
```

# LVS Reduction Priority

Specification statement

## LVS REDUCTION PRIORITY {PARALLEL | SERIES}

### Parameters

- **PARALLEL**

Keyword that specifies parallel reduction is preferred when devices can be reduced as either series or parallel. This is the default behavior if you do not include this statement in your rule file.

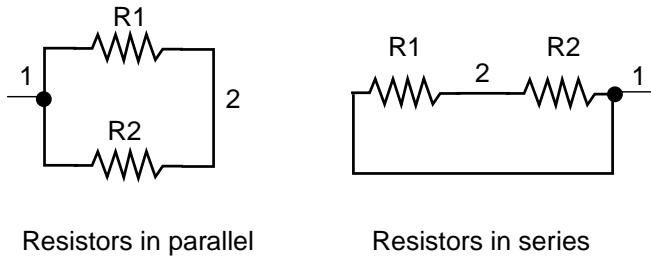
- **SERIES**

Keyword that specifies series reduction is preferred when devices can be reduced as either series or parallel.

### Description

Controls which type of device reduction to perform when both series and parallel reduction are possible on the same group of devices. Figure 4-163 shows an example of such devices:

**Figure 4-163. LVS Reduction Priority**



In this example, the resistors R1 and R2 can be considered in parallel between nets 1 and 2, or in series, with internal net 2. In general, two parallel devices can also be reduced in series when the following are all true:

- The two pins which form the parallel connection are swappable with each other.
- The two pins are not swappable with any other pins.
- At least one of the nets connected to the parallel devices has no other connections.

**Note**

 Depending on the options controlling filtering of unused devices, device pairs may be filtered as unused if they are reduced by one method but not by the other. For the example in [Figure 4-163](#), if LVS Filter Unused Option RB is specified, the resistors are considered unused when they are reduced in parallel. On the other hand, if LVS Filter Unused Option RC is specified, the resistors are considered unused when they are reduced in series.

---

If either series or parallel reduction (but not both) is possible, LVS reduces devices in accordance with the device reduction statements that appear in the rule file, as usual. If LVS Reduction Priority PARALLEL is in effect, either explicitly or by default, and parallel device reduction is disabled in an LVS Reduce Parallel statement, then the associated parallel devices are not reduced. Corresponding behavior applies if you specify LVS Reduction Priority SERIES.

If LVS cannot determine the topology of a set of devices, then they are not reduced and are treated as separate devices.

### Examples

```
// Reduce devices by series when they can be reduced by either series or
// parallel.
LVS REDUCTION PRIORITY SERIES
```

# LVS Report

Specification statement

## LVS REPORT *filename*

### Parameters

- *filename*

A required filename of the LVS report.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

### Description

Specifies the LVS report filename. This statement must be specified once in your rule file.

Naming convention for short isolation results — If LVS Report is used to supply a name, then setting [LVS Isolate Shorts](#) to YES produces the *short isolation results* filename:

<lvs-report-name>.shorts

Otherwise, the default filename for short isolation results is:

lvs.<layout-cell-name>.rep.shortcuts

You can specify the LVS Report in Calibre Interactive—nmLVS. See “[Specifying Outputs for an nmLVS Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In Pyxis Layout, this statement sets the value of application variable lvs\_default\_report\_name.

See also [LVS Report Maximum](#), [LVS Report Option](#), and [LVS Summary Report](#).

### Examples

```
// Specify LVS Report filename.  
LVS REPORT "lvs.rep"
```

# LVS Report Maximum

Specification statement

## LVS REPORT MAXIMUM {number | ALL}

### Parameters

- **number**

Positive integer that specifies an upper limit on the number of items printed in the [LVS Report](#). The default is 50 if you do not include this statement in the rule file.

- **ALL**

Keyword that specifies no upper limit on the number of items printed in the LVS report.

### Description

Specifies an upper limit on the number of items that appear in various sections of the LVS report. This limit affects the maximum number of discrepancies listed, the maximum number of items listed for each discrepancy, and the maximum number of items in various other lists printed by LVS, such as initial correspondence points and ambiguous devices or nets that are resolved arbitrarily. If the number of items for a given list is larger than the specified limit, the list is terminated.

Note that LVS usually lists the most informative (and most drastic) discrepancies first. Fixing the discrepancies that appear early in the list often clears later discrepancies. Therefore, using the default setting of 50, or a lower number, is most beneficial because such settings make the report easier to read.

Setting **number** to a high value or using ALL is rarely useful. However, if you want to see a complete listing of things like initial correspondence points or ambiguous nets and devices, then using settings such as these can be useful. Note that if your LVS run is incorrect, having a high value can make the report very long and difficult to use.

This statement can be specified once in your rule file.

You can specify the LVS Report maximum discrepancy count from the LVS Options pane in Calibre Interactive—nmLVS. See “[Specifying nmLVS Report Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

In Calibre PERC, this statement is overridden by the [PERC Report Maximum](#) setting.

In Pyxis Layout, this statement sets the value of application variable lvs\_report\_list\_limit.

### Examples

```
// Report a maximum of 25 discrepancies.  
LVS REPORT MAXIMUM 25
```

# LVS Report Option

Specification statement

**LVS REPORT OPTION** *option* [*option* ...]

## Summary

Controls LVS Report contents.

### Note



In general, you should accept the default LVS Report format and not specify this statement in your rule file. Some of these options cause the LVS Report to be long and difficult to read, and they should only be used to debug specific, known problems.

## Parameters

- *option*

A required, case-insensitive keyword that specifies the verbosity for a particular section of the LVS report or transcript. You may specify *option* any number of times in one statement, separated by spaces. The list of keywords is given in [Table 4-27](#).

**Table 4-27. Keywords for LVS Report Option**

Option	Description
A	<p>Detailed instance connections for INCORRECT nets.</p> <p>Enables the reporting of detailed instance connections on matched nets that are classified as INCORRECT (except for Short Circuit and Open Circuit discrepancies). Note that regardless of whether this option is specified, much of this information is also reported in a more concise form in the DETAILED INSTANCE CONNECTIONS section located at the bottom of the default LVS report. See <a href="#">Example 1</a>.</p>
AV	<p>Detailed instance connections for INCORRECT nets without correct devices report.</p> <p>Enables the same reporting as option A, with the exception that the AV option does not report correct devices on incorrect power and ground nets. For example, specifying this option does not generate the Correct Devices On This Net section in the LVS report when power or ground nets are reported as incorrect. Power and ground nets are defined by <a href="#">LVS Power Name</a> and <a href="#">LVS Ground Name</a> statements in the rule file. Specifying both the A and AV options generates the LVS report based only on the AV option.</p>
B	<p>Detailed instance connections in Short Circuit and Open Circuit discrepancies.</p> <p>Enables the reporting of detailed instance connections in Short Circuit and Open Circuit discrepancies. See <a href="#">Example 2</a>.</p>

**Table 4-27. Keywords for LVS Report Option (cont.)**

Option	Description
BV	<p>Detailed instance connections in Short Circuit and Open Circuit discrepancies without correct devices report.</p> <p>Enables the same reporting as option B, except for reporting correct devices on power and ground nets. In other words, if the net for which detailed connections are being reported is a power or ground net, correct instances are not reported. Power and ground nets are defined by <a href="#">LVS Power Name</a> and <a href="#">LVS Ground Name</a> statements in the rule file. Specifying both the B and BV options generates the LVS report based only on the BV option.</p>
BX	<p><a href="#">LVS Box</a> cells are listed in the setup section of the LVS Report. Cells are listed by layout and source, respectively. The cells listed are those specified in the rule file (all LVS Box statements are concatenated), whether they are present in the design or not.</p>
C	<p>Detailed instance connections in Missing Net and No Similar Net discrepancies.</p> <p>Enables the reporting of detailed instance connections in Missing Net and No Similar Net discrepancies. See <a href="#">Example 3</a>.</p>
CV	<p>Detailed instance connections in Missing Net and No Similar Net discrepancies without correct devices report.</p> <p>Enables the same reporting as option C, except for reporting correct devices on power and ground nets. In other words, if the net for which detailed connections are being reported is a power or ground net, correct instances are not reported. Power and ground nets are defined by LVS Power Name and LVS Ground Name statements in the rule file. Specifying both the C and CV options generates the LVS report based only on the CV option.</p>
D	<p>Detailed instance connections in Missing Instance and Missing Gate discrepancies.</p> <p>Enables the reporting of detailed instance connections reporting in Missing Instance and Missing Gate discrepancies. See <a href="#">Example 4</a>.</p>
E	<p>Ignore missing properties.</p> <p>Instructs LVS to ignore missing properties. Subsequently, missing properties are not reported and do not cause the overall result to be INCORRECT. Only the number of missing properties are reported in the statistics section. By default, missing properties are reported as discrepancies and cause the overall result to be INCORRECT. This option should be used with caution, as it disables a potentially important diagnostic.</p>

**Table 4-27. Keywords for LVS Report Option (cont.)**

Option	Description
E1	<p>Ignore missing properties only for LVS Reduce ... Tolerance.</p> <p>Instructs LVS to ignore missing properties that are used exclusively in TOLERANCE clauses of <a href="#">LVS Reduce</a> statements, and that are not used for any other purpose. Such missing properties are not reported as missing and do not cause the overall result to be INCORRECT. Only the number of missing properties is reported in the Statistics section.</p> <p>Missing properties that are used in other contexts (for example, in <a href="#">Trace Property</a> specification statements) continue to be reported as discrepancies and cause the overall result to be INCORRECT, even if they also appear in TOLERANCE clauses. (Unless such reporting is disabled by other options, such as option E.)</p> <p>As is normally the case, when a property used in a reduction TOLERANCE statement is missing on a device, then, with respect to that property, the device participates in the reduction as if the TOLERANCE statement were not present.</p>
F	<p>Disable smashed MOSFET warnings.</p> <p>Disables missing smashed MOSFET warnings, the unbalanced smashed MOSFET summary warning at the top of the LVS report, and respective warnings in the Information and Warnings section for the LVS report. Also see option O.</p>
FX	<p>Enable DETAILED ERROR ANALYSIS section of the report. This section can significantly improve the readability of the report and simplify debugging of numerous common LVS error types. For complete details of this section of the report, see “<a href="#">Detailed Error Analysis Reporting</a>” in the <i>Calibre Verification User’s Manual</i>.</p>
G	<p>Detailed instance connections in Property Error discrepancies.</p> <p>Enables the reporting of detailed instance connections in Property Error discrepancies.</p>
H	<p>Disable warning for cell correspondence names not found.</p> <p>Disables warnings in nmLVS-H about cell correspondence names not found in the layout or source. The tool normally prints these warnings in the transcript.</p>
I	<p>Disable reporting of isolated nets.</p> <p>Disables the reporting of isolated nets in the LVS report.</p>

**Table 4-27. Keywords for LVS Report Option (cont.)**

<b>Option</b>	<b>Description</b>
LPE	<p>Unmatched layout passthrough nets cause error.</p> <p>Causes the overall comparison result to be INCORRECT if there are <i>unmatched</i> passthrough nets present in the layout. By default, passthrough nets in the layout are reported as warnings in the INFORMATION AND WARNINGS section of the LVS report. Passthrough nets are nets that are connected to ports but not connected to anything else inside the subcircuit. Calibre nmLVS-H removes such nets and their ports unless they are named and can be used as initial correspondence points. For a full discussion and examples, see the section “<a href="#">Passthrough Nets</a>” in the <i>Calibre Verification User’s Manual</i>. See also options SP and SPE.</p>
MC	<p>Device instance counts based on model name.</p> <p>Controls the reporting of device instance counts based upon model name (subtype). By default, initial instance counts and instance counts after transformations are reported per element name (component type). For example, one count is reported for all MN devices, even if there are MN devices with different model names. When MC is specified, instance counts are reported separately for each element name and model name. See <a href="#">Example 5</a>.</p>
N	<p>Report missing layout names as errors.</p> <p>Instructs LVS to report layout names that are missing in the source as errors. LVS also reports the overall comparison result as INCORRECT and generates the secondary comparison status message:</p> <pre>Error: Layout names missing in source (see Information and Warnings).</pre> <p>The following header is included in the INFORMATION AND WARNINGS section of the LVS report:</p> <pre>Error: Layout Names That Are Missing In The Source:</pre>
NCA	<p>Suspend nmLVS-H comparison upon detection of a NOT COMPARED cell status.</p> <p>If a cell with the status of NOT COMPARED is detected, all remaining cells that have not been fully compared are also reported as NOT COMPARED (no attempt is made to finish comparing the remaining cells) and the run terminates with a NOT COMPARED status. Cells that may have been compared as either CORRECT or INCORRECT prior to the detection of the first NOT COMPARED cell retain their compared statuses.</p>

**Table 4-27. Keywords for LVS Report Option (cont.)**

<b>Option</b>	<b>Description</b>
NE	Mismatched instance counts in corresponding cells triggers NOT COMPARED status.  Instructs LVS to report a NOT COMPARED status for corresponding cells that have differing instance counts in layout and source. Both instance type (element) and subtype (model) are checked when comparing instance counts. In flat LVS, this option causes the top-level comparison status to be NOT COMPARED if the instance counts do not match between layout and source.
NOK	Comparison results for incorrect hcells only.  Instructs nmLVS-H to provide detailed comparison results only for hcells in which discrepancies occur. By default in nmLVS-H, all hcells are reported in detail whether they compare correctly or not. If you specify this option, only the details of hcells having INCORRECT results are reported; the overall results of all cells are still reported in the OVERALL COMPARISON RESULTS section. When this option is active, the CELL COMPARISON RESULTS section header changes to CELL COMPARISON RESULTS ( INCORRECT CELLS ONLY ).
O	Status message for ambiguity resolution and smashed MOSFETs.  Enables reporting of ambiguity resolution and unbalanced smashed MOSFETs status messages in the OVERALL COMPARISON RESULTS and CELL COMPARISON RESULTS sections. This reporting would be otherwise disabled by LVS Report Option options F, R, or RA. If LVS Report Option O is specified, options F, R, and RA disable reporting of ambiguity resolution points and unbalanced smashed MOSFETs, respectively, but the overall status messages are printed followed by the string “(reporting disabled)”.
P	Disable warnings for direct connections between different ports.  Disables the reporting of “Direct connections between different ports” warnings in the connectivity extractor.
PG	Always report hcells with supply mismatch.  Causes nmLVS-H to report all hcells that have mismatched power and ground nets. Calibre nmLVS-H reports the comparison status as NOT COMPARED for hcells that have different supply nets in layout and source (for example, power net is missing in layout). By default, however, this status is reported only if logic gate recognition is enabled, and at least one gate is recognized in the cell in either layout or source. The PG option ignores the dependency on logic gate recognition and reports any hcells with power and ground net mismatches.

**Table 4-27. Keywords for LVS Report Option (cont.)**

Option	Description						
R	<p>Disable ambiguity resolution report for correct results.</p> <p>Disables reporting of ambiguity resolution points in LVS when the overall comparison result is CORRECT. This option controls reporting of ambiguity resolution points in the Information and Warnings section of the LVS report, as well as reporting of ambiguity resolution status messages in the OVERALL COMPARISON RESULTS and CELL COMPARISON RESULTS sections of the LVS report. When the overall comparison result is other than CORRECT, ambiguity resolution points are reported as usual, regardless of this option. Also see options O and RA.</p>						
RA	<p>Disable ambiguity resolution report for all results.</p> <p>Disables reporting of ambiguity resolution points in LVS independent of the overall comparison result. This option is similar to LVS Report Option R, which also disables reporting of ambiguity resolution points (but only if the overall comparison result is CORRECT). Both options control reporting of ambiguity resolution points in the INFORMATION AND WARNINGS section of the LVS report.</p>						
RD	<p>Marks reduced instances in the LVS report.</p> <p>If LVS Report Option RD is specified, names of reduced instances in the LVS report are followed by the string “(SL)” for layout instances or “(SS)” for source instances. Note that these annotations are the same as the ones used in instance cross-reference files to mark reduced instances.</p> <p>For example, a property error report involving a reduced layout instance may appear as follows:</p> <table style="width: 100%; text-align: center;"> <tr> <td style="width: 33%;">1      m2 (SL)    MP(P)</td> <td style="width: 33%;">m1    MP(P)</td> <td style="width: 33%;">100%</td> </tr> <tr> <td>w: 2 u</td> <td>w: 1 u</td> <td></td> </tr> </table>	1      m2 (SL)    MP(P)	m1    MP(P)	100%	w: 2 u	w: 1 u	
1      m2 (SL)    MP(P)	m1    MP(P)	100%					
w: 2 u	w: 1 u						
S	<p>Detailed reports of Sconnect conflicts.</p> <p>Enables the detailed reporting of <b>Sconnect</b> conflicts in the LVS run transcript and the circuit extraction report, when generated. The report format is discussed under “Connectivity Extraction Errors and Warnings” in the <i>Calibre Verification User’s Manual</i> or the <i>ICVerify User’s and Reference Manual</i>.</p>						

**Table 4-27. Keywords for LVS Report Option (cont.)**

Option	Description
SP	<p>Report passthrough nets in source.</p> <p>Enables reporting of passthrough nets in the source. Such nets are reported in the INFORMATION AND WARNINGS section of the LVS Report as warnings. Passthrough nets are nets that are connected to ports but not connected to anything else inside the subcircuit. Calibre nmLVS removes such nets and their ports, unless they are named and can be used as initial correspondence points. For a full discussion and examples, see the section “<a href="#">Passthrough Nets</a>” in the <i>Calibre Verification User’s Manual</i>. See also option SPE.</p>
SPE	<p>Unmatched source passthrough nets cause error.</p> <p>Causes the overall comparison results to be INCORRECT if there are <i>unmatched</i> passthrough nets in the source. The nets are reported as for option SP. For a full discussion and examples, see the section “<a href="#">Passthrough Nets</a>” in the <i>Calibre Verification User’s Manual</i>. See also option LPE.</p>
V	<p>Report virtual connections.</p> <p>Instructs the connectivity extractor in Calibre and ICVerify to report virtual connections (for example, from the <a href="#">Virtual Connect Name</a> or <a href="#">Virtual Connect Colon YES</a> specification statements). Virtual connections are reported as notes in the connectivity extraction report. Specifying this option is equivalent to specifying <a href="#">Virtual Connect Report YES</a>.</p>
W	<p>Error when source and layout refer to same data.</p> <p>Instructs LVS to report an error, rather than a warning, when source and layout refer to the same data. When the source and layout refer to the same data, this option causes the overall comparison result to be INCORRECT.</p>
X	<p>Instance list for COMPONENT TYPES WITH NON-IDENTICAL SIGNAL PINS.</p> <p>Instructs nmLVS-H to provide a list of representative instances for each configuration of pins reported in the COMPONENT TYPES WITH NON-IDENTICAL SIGNAL PINS section of the report. Each list of instances is preceded by its parent cell name. The final pin configuration (after discarding of extra pins) is shown as well. See the section “Component Types with Non-Identical Signal Pins” in the <i>Calibre Verification User’s Manual</i>.</p>

**Table 4-27. Keywords for LVS Report Option (cont.)**

Option	Description
XR	<p>Disable creation of cross-reference database (XDB) when comparison result is CORRECT.</p> <p>By default, creation of the cross-reference database is controlled by options of the <a href="#">Mask SVDB Directory</a> specification statement, as well as command-line arguments -ixf and -nxm. If option XR is specified, then the database is not created if there are no LVS errors and the overall comparison status is CORRECT. This option can be useful if the cross-reference database is used only to debug LVS errors and generating it takes considerable time. This option has no effect if the cross-reference database is not requested.</p>

## Description

Controls the level of detail and verbosity of an [LVS Report](#). This is an optional statement. The default report format is *concise*, that is, no report options are set. You should accept the default unless you have specific reasons not to.

Statements can be repeated and options are accumulated.

You can specify LVS Report options from the LVS Options pane in Calibre Interactive—nmLVS. See “[Specifying nmLVS Report Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## Examples

### Example 1

This example illustrates the default output followed by output of LVS Report Option A:

Default

```
5 Net 8(800,500)          DOG
-----
** unmatched connection ** X1/M5:D
53(530,500):D           ** unmatched connection **
```

Option A specified

```
5 Net 8(800,500)          DOG
-----
--- Incorrect Devices On This Net ---

54(540,500) MP(PA)        X1/M1  MP(PA)
g: 1(100,500)             g: A
s: 2(200,500)             s: B
** 8(800,500) **          d: DOG
d: 7(700, 500)            ** CAT **

53(530,500) MP(PA)        X1/M4  MP(PA)
g: 1(100,500)             g: A
```

```

s: 2(200,500)          s: B
d: 8(800,500)          ** DOG **
** 6(600,500) **       d: PIG

--- Correct Devices On This Net ---

55(550,500) MP(PA)      X1/M5  MP(PA)
g: 8(800,500)           g: DOG
s: 2(200,500)           s: B
d: 4(400,500)           d: D

56(560,500) MP(PA)      X1/M5  MP(PA)
g: 8(800,500)           g: DOG
s: 2(200,500)           s: B
d: 4(400,500)           d: D

```

**Example 2**

This example illustrates the default output followed by output of LVS Report Option B:

**Default**

```
5 Net 7(700,500)          CAT
                           DOG
```

**Option B specified**

```
5 Net 7(700,500)          CAT
                           DOG
```

--- Devices on source net DOG ---

```
52(520,500) MP(PA)      X1/M3  MP(PA)
g: 7(700,500)           g: DOG
s: 2(200,500)           s: B
d: 4(400,500)           d: D
```

--- Devices on source net CAT ---

```
53(530,500) MP(PA)      X1/M4  MP(PA)
g: 7(700,500)           g: CAT
s: 2(200,500)           s: B
d: 4(400,500)           d: D
```

```
54(540,500) MP(PA)      X1/M5  MP(PA)
g: 7(700,500)           g: CAT
s: 2(200,500)           s: B
d: 4(400,500)           d: D
```

Note: The smaller net appears first.

**Example 3**

This example illustrates the default output followed by output of LVS Report Option C:

**Default**

```
5 7(700,500)          ** missing net **
```

**Option C specified**

## LVS Report Option

---

```
5 7(700,500)          ** missing net **  
--- Devices on layout net 7(700,500) ---  
  
50(500,500) MP(PA)      X1/M1 MP(PA)  
g: 1(100,500)           g: A  
s: 2(200,500)           s: B  
d: 7(700,500)           ** missing net **  
** 3(300,500) **       d: C  
  
51(510,500) MP(PB)      X1/M2 MP(PB)  
g: 1(100,500)           g: A  
s: 2(200,500)           s: B  
d: 7(700,500)           ** missing net **  
** 3(300,500) **       d: C
```

### Example 4

This example illustrates the default output followed by output of LVS Report Option D:

#### Default

```
5 52(520,500)          ** missing instance **
```

#### Option D specified

```
5 52(520,500) MP(PA)      ** missing instance **  
g: 7(700,500)           ** DOG **  
s: 2(200,500)           ** B **  
d: 4(400,500)           ** D **
```

### Example 5

This example illustrates the default output followed by output of LVS Report Option MC:

#### Default

```
INITIAL NUMBERS OF OBJECTS  
-----  
Layout      Source      Component Type  
-----      -----  
Instances:   20        30        *      mn (4 pins): g (s d) b
```

```
NUMBERS OF OBJECTS AFTER TRANSFORMATION  
-----  
Layout      Source      Component Type  
-----      -----  
Instances:   10        11        *      mn (4 pins): g (s d) b
```

It may be difficult to determine why there is an extra device in the source. With LVS Report Option MC, the counts are reported separately for all MN devices with different model names:

## Option MC specified

INITIAL NUMBERS OF OBJECTS

	Layout	Source	Component Type
Instances:	-----	-----	-----
	20	15	* mn(N) (4 pins): g (s d) b
	0	15	* mn() (4 pins): g (s d) b

NUMBERS OF OBJECTS AFTER TRANSFORMATION

	Layout	Source	Component Type
Instances:	-----	-----	-----
	10	10	* mn(N) (4 pins): g (s d) b
	0	1	* mn() (4 pins): g (s d) b

Now it is clear that the extra device in the source has a different model name. It is likely that this device was not reduced together with other MN devices, because device reduction operates only on devices with identical element and model names.

# LVS Report Units

Specification statement

## LVS REPORT UNITS {YES | NO}

### Parameters

- **YES**

Keyword specifying that built-in device properties appear with units, and that values of such properties are appropriately scaled prior to reporting in order to match a predefined set of reporting units. This is the default behavior.

- **NO**

Keyword that specifies properties are reported without units and that no scaling occurs.

### Description

Indicates whether certain built-in device properties should appear with or without units in the LVS report. Examples of properties affected by this statement are: L, W, AD, AS, PD, PS, A, P, C, and R. The NO setting specifies that all property values should be reported as unitless numbers and that no scaling should occur during reporting.

This statement may be specified once.

### Examples

#### Example 1

This statement is especially useful when you want to express property values in source and layout netlists with unitless numbers or in *user units*. This can be achieved as follows:

```
UNIT LENGTH 1
LVS REPORT UNITS NO
```

For instance, given those setting and the following source netlist:

```
M0 3 1 2 4 P L=11 W=11 AD=101 AS=101
```

LVS could report the following property errors:

w: 10	w: 11	9.09%
ad: 100	ad: 101	0.99%

Note that values are reported as they appear in the netlist. With the default setting LVS Report Units YES, LVS reports the same errors as follows (which would be misleading):

w: 10 m	w: 11 m	9.09%
ad: 100 sq m	ad: 101 sq m	0.99%

**Example 2**

Consider a rule file with the following setting:

```
UNIT CAPACITANCE 1e-15
```

performing netlist-to-netlist comparison between these two netlists:

Layout: C1 5 6 C=5

Source: C1 5 6 C=7

With LVS Report Units NO, LVS reports this property error:

c: 5	c: 7	28.6%
------	------	-------

Again, values are reported as they appear in the netlist. With LVS Report Units YES, LVS would report the same error as follows:

c: 5e+15 ff	c: 7e+15 ff	28.6%
-------------	-------------	-------

## LVS Report Warnings Hcell Only

Specification statement

### LVS REPORT WARNINGS HCELL ONLY {NO | YES}

Used only in Calibre nmLVS-H and PERC.

#### Parameters

- **NO**

Keyword that specifies connectivity extraction warnings are not filtered by hcells. This is the default behavior.

- **YES**

Keyword that specifies connectivity extraction warnings are reported only for hcells.

#### Description

Indicates whether the hierarchical circuit extraction report and transcript contain warnings from cells specified as hcells only. (The top-level cell is always considered an hcell.) Hcells are specified using either the [Hcell](#) statement or through passing a file containing hcells using the -hcell switch at invocation. The default is NO, which means warnings from all levels are reported.

If YES is specified, both the circuit extraction report and transcript do not contain any warnings for cells that are not in the hcell list. Suppressed warnings include short-circuit, open-circuit, ambiguously-named top-level ports, and unattached-name top-level ports.

---

**Note**



The -automatch command line option does not apply to this statement. Automatch is not applied during the connectivity extraction module.

---

This statement may be specified once. If both LVS Report Warnings Hcell Only YES and [LVS Report Warnings Top Only](#) YES are specified, warnings are reported only for the top level cell.

This statement has no effect on the contents of the extracted layout netlist. The YES option can suppress warnings during connectivity extraction in non-hcells, but this does not prevent discrepancies related to suppressed warnings from being detected during the LVS comparison stage. If LVS comparison reveals a mismatch that is difficult to debug, then specifying NO may help to resolve the problem in the connectivity extraction stage where the warnings would be of benefit.

#### Examples

```
// Report connectivity extraction warnings from all cells.  
LVS REPORT WARNINGS HCELL ONLY NO
```

# LVS Report Warnings Top Only

Specification statement

## LVS REPORT WARNINGS TOP ONLY {NO | YES}

Used only in Calibre nmLVS-H.

### Parameters

- **NO**  
Keyword that specifies connectivity extraction warnings are reported from all levels of the design. This is the default behavior.
- **YES**  
Keyword that specifies connectivity extraction warnings are reported only for the top-level cell.

### Description

Indicates whether the hierarchical circuit extraction report and transcript contain warnings about the top-level cell only. The default is NO, which means warnings from all levels are reported.

If YES is specified, both the circuit extraction report and transcript do not contain any warnings for lower-level cells. Suppressed warnings include short-circuit, open-circuit, ambiguously-named top-level ports, and top-level ports with unattached names.

This statement may be specified once. This statement has no effect upon the contents of the extracted layout netlist; hence, discrepancies related to suppressed warnings can be detected in a subsequent LVS comparison run.

#### Caution



Use of this statement can suppress the reporting of legitimate problems at subcell levels.  
Be careful about specifying YES in a tapeout run.

See also [LVS Report Warnings Hcell Only](#).

### Examples

```
// Report connectivity extraction warnings from all cell levels.
LVS REPORT WARNINGS TOP ONLY NO
```

## LVS Reverse WL

Specification statement

### LVS REVERSE WL {NO | YES}

#### Parameters

- **NO**

Keyword that instructs LVS not to reverse L and W values not prefixed with explicit L= and W= designators. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs LVS to reverse L and W values not prefixed with explicit L= and W= designators.

#### Description

Specifies to reverse the default L,W order in MOS devices in SPICE netlists. The default is NO. YES specifies that the order is W,L.

This statement also reverses the default L,W order within SPICE-like properties in EDDM and Direct Mode ICverify designs and is similar to the SPICE statement .OPTION WL.

#### Examples

##### Example 1

```
LVS REVERSE WL NO // do not switch parameter order
```

##### Example 2

This example reverses the order of the L and W values for the following MOS device:

```
M1 d g s b pmos 1 6
```

Normally, L = 1 and W = 6. However, these values are reversed to L = 6 and W = 1 when the following statement is specified:

```
LVS REVERSE WL YES
```

# LVS Short Equivalent Nodes

Specification statement

## LVS SHORT EQUIVALENT NODES {NO | **SPLIT**}

### Parameters

- **NO**

Keyword that instructs the tool not to short equipotential nodes together in split gate MOSFET structures. This is the default behavior.

- **SPLIT**

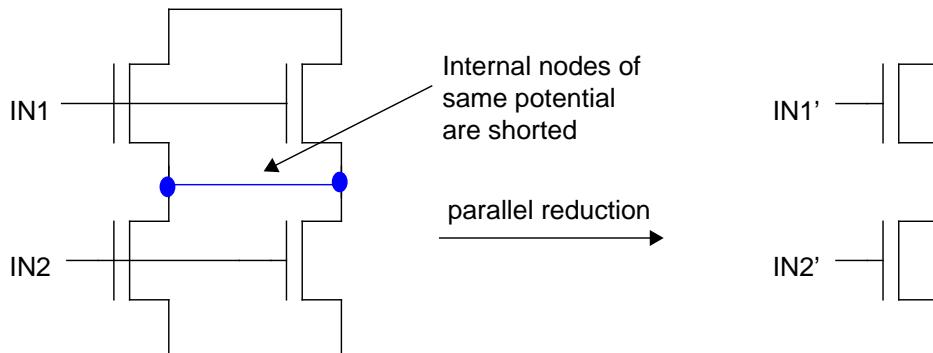
Keyword that instructs the tool to short together equipotential nodes in split gate MOSFET structures.

### Description

Specifies whether to short together the internal nets of basic split gate structures if the nodes are equipotential. For detailed information about split-gate reduction, refer to “Split Gate Reduction” in the *Calibre Verification User’s Manual*.

If SPLIT is specified, then the internal equipotential nets of basic split gate MOSFET structures are shorted together and replaced by a single net. Note that more complex variants of split gate structures such as semi-split gates or series-parallel split gates *are not performed* by the SPLIT option.

**Figure 4-164. SPLIT Option Behavior**



Once the equipotential nets are shorted, the split gate structure becomes a series of groups of parallel MOS devices. These groups can be further reduced by parallel reduction controlled by the [LVS Reduce Parallel MOS YES](#) or the [LVS Reduce](#) equivalent specification statements.

The [LVS Reduce Split Gates YES](#) specification statement (the default) is incompatible with the LVS Short Equivalent Nodes SPLIT option. If both statements are present in the same rule file, an LSEN1 compiler error is issued.

### Examples

#### Example 1

This is a typical usage of the default.

```
// Do not short equivalent split gate nodes
LVS SHORT EQUIVALANT NODES NO

LVS REDUCE SPLIT GATES YES
LVS REDUCE PARALLEL MOS YES
```

#### Example 2

This is for shorting equivalent nodes in simple split-gate structures.

```
//Short equivalent split gate nodes
LVS SHORT EQUIVALANT NODES SPLIT

LVS REDUCE SPLIT GATES NO
LVS REDUCE PARALLEL MOS YES
```

# LVS Show Seed Promotions

Specification statement

## LVS SHOW SEED PROMOTIONS {NO | YES}

Used only in Calibre nmLVS-H and PERC

### Parameters

- **NO**

Keyword that instructs the tool not to create a file that lists instances of seed promotion. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to create a file that lists instances of seed promotion. [LVS Push Devices](#) YES (the default) disables LVS Show Seed Promotions YES. LVS Push Devices NO is required for seed promotion to be shown.

### Description

---

**Note**



In general, do not specify YES. This setting may dramatically increase run time and it disables multi-threaded device recognition. Use YES only as a diagnostic tool to find undesirable seed promotions.

---

Specifies whether instances in the design where devices are formed by interaction of polygons from different levels of hierarchy are reported. This statement may appear once.

This statement operates in Calibre applications that perform hierarchical circuit extraction. It finds places in the design where devices are formed by interaction of polygons from different levels of hierarchy. For example:

- The device recognition polygon (seed polygon) can be in a lower-level cell with one of the pin polygons on a higher level.
- Two seed polygons from different cell placements may overlap, forming one device at a higher level.

These devices are formed in a higher level cell—a cell from which all polygons required to form the device are visible (the polygons are present either in the cell itself or in its sub-hierarchy). This cell is the lowest common ancestor of all cell placements that form the device. This process is known as *seed promotion*.

Seed promotion allows Calibre to recognize devices that are formed across levels of hierarchy at the price of some degradation in the original design hierarchy. For example, in general, cells where seed promotion occurs cannot be used as corresponding cells (hcells) in hierarchical LVS.

The nmLVS hierarchical SPICE netlister places \*.SEEDPROM statements in all subcircuits that were subject to seed promotion. LVS Show Seed Promotions YES gives detailed information about seed promotions.

When LVS Show Seed Promotion YES is indicated, the file:

```
<layout_primary>.seed_promotion
```

is created in the [Mask SVDB Directory](#), if specified, or otherwise in the current directory. The file contains results in nmDRC ASCII form (see [DRC Results Database](#)). It consists of one or more checks each containing one or more polygons. Each check has a name consisting of four parts separated by spaces:

```
<seed_cell> <seed_layer> <interacting_layer> <interacting_cell>
```

where:

- <seed\_cell> is the name of a hierarchical cell from which devices were promoted. The seed polygons of the promoted devices may be located in that cell or in its sub-hierarchy.
- <seed\_layer> is the name of a [Device](#) statement seed layer on which the seed polygons reside.
- <interacting\_layer> is the name of a layer that is either the seed layer or one of the pin or auxiliary layers associated with the seed layer.
- <interacting\_cell> is the name of a hierarchical cell in which interaction between the <seed\_layer> and the <interacting\_layer> has caused the polygons of the check to be promoted from the <seed\_cell> to the <interacting\_cell>.

For example:

```
MUX_03 RES MET1 BLOCK_6
```

Each check contains a four line comment explaining the check. The comment for the check in the example above would be:

```
Device seed originating in or below  
cell "MUX_03" on layer "RES"  
and promoted due to interaction  
with layer "MET1" in cell "BLOCK_6".
```

All polygons are in the coordinate system of the top level cell of the design, no matter what the <seed\_cell> or <interacting\_cell>. They have been flattened into the top-level cell and must be viewed as checks in that cell.

Redundant shapes are suppressed. The following example illustrates what this means:

Suppose the top-level cell contains 1,000 instances of cell B. Cell B contains an instance of cell A. A seed in cell A is promoted to cell B due to touching by a shape in cell B. In this instance, seed promotion is due to an interaction between cell B and cell A. The system only shows one such seed shape flattened into the top-level cell rather than 1,000 shapes (one for each instance of cell B).

Another example:

Suppose that the top-level cell from the previous example also contains a shape on the touching layer that touches the seed in cell A in only one of the 1,000 instances of cell B. In this case, the system produces two flattened seed shapes in the check. One for the interaction between cell B and cell A, and a second for the interaction between the top-level cell and cell A. These seed shapes may, in fact, be the same shape, but it is reported once for each touch because the touches took place in separate cells.

If a seed is touched by multiple shapes on the same touching layer in a given cell, only one check is generated. It is only in the case where touches take place in different cells that more than one check is generated.

If an [Hcell](#) list has been provided in the run, then promotions where the <seed\_cell> is *not* an hcell are suppressed. This is useful because such promotions would not have an effect on circuit comparison in hierarchical LVS.

Note that seed promotion occurs in the hierarchical device recognition stage and deals with interactions between layers that appear in Device operations. It does not deal with geometry promotion that occurs during the derivation of those layers (for example, in Boolean operations). So, LVS Show Seed Promotions does not isolate such promotion.

Seed promotions are also reported in the session transcript, regardless of whether or not LVS Show Seed Promotions is indicated.

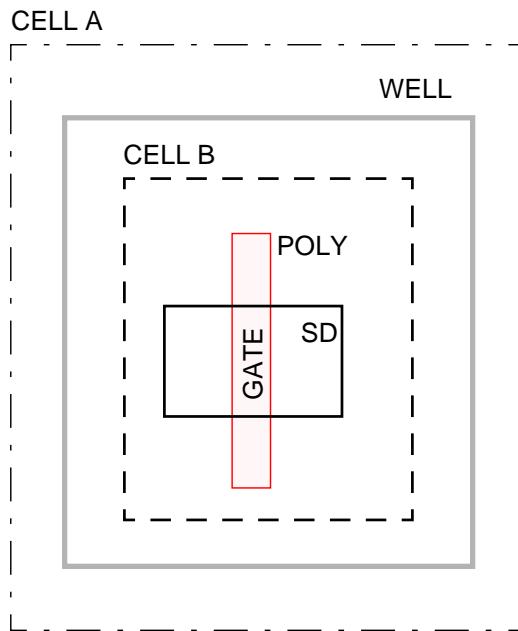
You can specify seed promotion reporting behavior from the LVS Options pane in Calibre Interactive—nmLVS. See “[Specifying nmLVS Report Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [LVS Show Seed Promotions Maximum](#) and [LVS Expand Seed Promotions](#).

## Examples

In [Figure 4-165](#), GATE is the seed layer; POLY, SD, and WELL are pin layers. The seed polygon on layer GATE is formed in cell B. However, the WELL polygon is higher up in cell A. There is no WELL polygon in cell B.

**Figure 4-165. Seed Promotion**



The rule file entries are as follows:

```
SD = DIFF NOT POLY
GATE = POLY AND DIFF
CONNECT METAL POLY SD BY CONT
DEVICE MP GATE POLY SD SD WELL
```

This device, formed by interaction of polygons from different levels of hierarchy, is promoted to cell A and becomes part of that cell. The statement LVS Show Seed Promotions YES shows the GATE polygon in the coordinate space of the top level cell. The name of the check is:

```
B GATE WELL A
```

which indicates that the seed polygon on layer GATE was promoted from cell B because of interaction with layer WELL which occurred in cell A.

## LVS Show Seed Promotions Maximum

Specification statement

### LVS SHOW SEED PROMOTIONS MAXIMUM *number*

Used only in Calibre nmLVS-H and PERC

#### Parameters

- *number*

A required positive integer that specifies the maximum number of polygons to appear in any LVS Show Seed Promotions check.

#### Description

Specifies the maximum number of polygons that appear in any check performed by the LVS Show Seed Promotions statement. This statement may appear once. There is no way to limit the number of checks.

Refer to [LVS Show Seed Promotions](#) for more information on the checks.

You can specify seed promotion reporting behavior from the LVS Options pane in Calibre Interactive—nmLVS. See “[Specifying nmLVS Report Options](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

#### Examples

```
// Show seed promotion data. Use this only when it's needed for debug.  
LVS SHOW SEED PROMOTIONS YES  
  
// Show up to 50 seed promotion polygons.  
LVS SHOW SEED PROMOTIONS MAXIMUM 50
```

# LVS Signature Maximum

Specification statement

**LVS SIGNATURE MAXIMUM {*number* | ALL}**

## Parameters

- ***number***

Integer greater than or equal to one, that specifies maximum neighborhood size. This represents a number of nodes.

- **ALL**

Keyword that specifies unlimited neighborhood size.

## Description

Specifies that LVS computes signatures for neighborhoods no bigger than a specified maximum size. This statement is only used by the LVS comparison module.

In LVS, a *neighborhood* refers to the instances and nets around an object of no more than *n* nodes away. The distance between an instance and an adjacent net is 1 by default. A *signature* is data that represents information about an object and its neighborhood in a concise form.

Normally, LVS computes maximum neighborhood sizes dynamically using the design size and other factors. This statement overrides this behavior and specifies a maximum neighborhood size larger (or smaller) than would otherwise be calculated.

---

### Note



Specifying a larger neighborhood may eliminate ambiguity but with potentially longer run time.

---

For more information about this process, refer to section “Matching of Circuit Elements” in the *Calibre Verification User’s Manual*.

## Examples

```
// Use a neighborhood size of 10.  
LVS SIGNATURE MAXIMUM 10
```

# LVS Soft Substrate Pins

Specification statement

## LVS SOFT SUBSTRATE PINS {NO | YES}

### Parameters

- **NO**

Keyword that instructs LVS to treat substrate and bulk pins like any other pins. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs LVS to treat substrate and bulk pins with less importance in circuit comparison.

### Description

Specifies whether substrate and bulk pins have a standard effect on how circuit elements are matched.

NO (the default) indicates that substrate and bulk pins should be treated like any other pins.

YES indicates that substrate and bulk pins are treated with less importance in circuit comparison. In other words, they have little effect on how circuit elements are matched. This helps the LVS circuit comparison when major discrepancies in substrate or bulk connections are expected. Their connectivity is checked for correctness at the end of processing but they have little effect on the matching operation itself. Discrepancies involving substrate pin connections appear in a separate section of the LVS report.

YES is recommended when a significant number of devices have substrate or bulk connections that are different in layout and source. For example, when you have several different ground supplies connected to the same substrate region in the layout (typically the chip extent minus wells) and you use the [Sconnect](#) operation to connect substrate to ground.

Substrate and bulk pins are pin B in MOS devices, pin S in Q devices, the fourth and subsequent pins of J devices, the third and subsequent pins of L and V devices, and any other optional pins in MOS, Q, R, C and D devices. MOS devices are component types MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD and LDD. Devices must be built-in for comparison in order to participate. See the [Device](#) section for details.

Note that this statement does not control the LVS transformation phase (such as, series and parallel device reduction, logic gate recognition, unused device filtering).

### Examples

```
// Substrate and bulk pins are treated as any other pins.
LVS SOFT SUBSTRATE PINS NO
```

## LVS Softchk

Specification statement

**LVS SOFTCHK** *lower\_layer* [LOWER | CONTACT | UPPER] [ALL]  
[DIRECT] [MASK]

### Parameters

- ***lower\_layer***  
A required original layer or layer set, or a derived polygon layer.
- **LOWER | CONTACT | UPPER**  
An optional keyword set that selects polygons involved in conflicting connections from the specified layer. Possible choices are:
  - LOWER** — Selects *lower\_layer* polygons from an [Sconnect](#) operation. This is the default behavior if you do not include a selection from this set in the statement.
  - CONTACT** — Selects *contact\_layer* polygons from an Sconnect operation.
  - UPPER** — Selects *upper\_layer* polygons from an Sconnect operation.For Sconnect operations without the BY keyword, CONTACT and UPPER are synonymous.
- **ALL**  
An optional keyword that specifies all electrical nodes involved in conflicting connections to the *lower\_layer* polygon are eligible for reporting. By default, the node that Sconnect chooses as the intended connection is not reported. The ALL keyword has no effect if you specify LOWER.
- **DIRECT**  
An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.
- **MASK**  
An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

### Description

Selects *lower\_layer*, *upper\_layer*, or *contact\_layer* polygons, depending on what you specify, from all [Sconnect](#) operations involved in conflicting connections to the specified *lower\_layer*. The *lower\_layer* parameter must appear as a *lower\_layer* argument in one or more Sconnect operations. By default, for a particular *lower\_layer* polygon, all nodes that form soft connections to that polygon are reported, except the node chosen by Sconnect as the non-

conflicting connection. If there are no Sconnect conflicts, then nothing is reported. The ALL keyword shows all connections to the *lower\_layer* polygon.

LVS Softchk observes the [ERC Maximum Vertex](#) specification statement. Large LVS Softchk result polygons are segmented according to the ERC Maximum Vertex value before being written to the LVS Softchk results database. If ERC Maximum Vertex is not specified, then the ERC Maximum Vertex default of 4096 is used.

For hierarchical and flat Calibre nmLVS, LVS Softchk results are written out in nmDRC ASCII results database format in the [Mask SVDB Directory](#) to the file *layout\_primary*.softchk where *layout\_primary* is the top-level layout cell name. If Mask SVDB Directory is not specified, the results file *layout\_primary*.softchk is written to the current directory.

This file contains results in the coordinate space of the top-level cell. In hierarchical operation, results from lower-level cells are shown only once; the lowest leftmost placement of a cell is used as representative for Sconnect conflicts occurring in that cell. This data reduction is done per net, not per polygon. All polygons of the net are output if a net is selected for output.

The result file contains one nmDRC check for each non-empty Softchk operation. Check names are constructed from the respective Softchk operation arguments and consist of the word SOFTCHK followed by the *lower\_layer* name, the words CONTACT, UPPER or LOWER, and the keyword ALL if indicated. For example, for the operation:

```
LVS SOFTCHK NWELL CONTACT
```

the check name is:

```
SOFTCHK NWELL CONTACT
```

The LVS Softchk operation does not report polygons from Sconnect operations that appear in early stages of an Sconnect chain. Refer to section “[Chained Sconnect Operations](#)” on page 1648 for more information about Sconnect chains.

To view conflicts involving Sconnect operations from multiple stages of the chain, use layer operations as shown in the following example:

Given the rule file operations:

```
CONNECT A AA
SCONNECT A B
SCONNECT B C
```

and that you want to view all polygons on layer A that failed to stamp layer C, through the intermediate step of stamping layer B, LVS Softchk can only report polygons on layer A that were rejected when stamping layer B. To get the desired results, use the following rule check and ERC operations.

```
REJECTED_A {
    A NOT OUTSIDE (EXTERNAL C A == 0 INSIDE ALSO REGION NOT CONNECTED) }

ERC RESULTS DATABASE ercddb
ERC SELECT CHECK REJECTED_A
```

LVS Softchk operations are executed in LVS after ERC checks and [ERC Pathchk](#) operations. They are not executed in nmDRC or Direct-mode ICverify applications.

You can check for soft connections in nmDRC applications with the [External](#) operation. Examples are shown in the “Soft Connection Checks” section of the [Calibre Verification User’s Manual](#).

See “[LVS Softchk Debug Method](#)” in the *Calibre Verification User’s Manual* for more details on how to debug LVS Softchk errors.

In a stamping conflict, the Sconnect operation uses vertex count of an *upper\_layer* net to determine the net ID that a *lower\_layer* polygon receives. In some cases, it may be more useful to use the number of contacts shared between *upper\_layer* and *lower\_layer* polygons to resolve a stamping conflict. To use this method, see “[Reporting Soft Connections Using Contact Counts](#)” in the *Calibre Verification User’s Manual*.

For ICtrace(M), LVS Softchk results are written in the current directory to the file *layout\_primary.softchk*, where *layout\_primary* is the top-level layout cell name (if [Layout Primary](#) is specified). If Layout Primary is not specified (implying the database is not geometric) the file name is *lvs.softchk*.

### Examples

```
SCONNECT PTAP PWELL
LVS SOFTCHK PWELL UPPER // show taps that cause stamping conflict
LVS SOFTCHK PWELL LOWER // show bad pwells
```

To show ptap polygons on all the nets involved in a stamping conflict, specify this:

```
LVS SOFTCHK PWELL UPPER ALL
```

# LVS Spice Allow Duplicate Subcircuit Names

Specification statement

## LVS SPICE ALLOW DUPLICATE SUBCIRCUIT NAMES {YES | NO}

### Parameters

- YES

Keyword that instructs the tool to issue a warning when it detects a duplicate subcircuit definition where the name and pin count match another subcircuit in the SPICE netlist. This is the default behavior if you do not specify this statement.

- NO

Keyword that instructs the tool to issue a fatal error when it detects a duplicate subcircuit definition where the name and pin count match another subcircuit in the SPICE netlist.

### Description

Specifies whether a SPICE netlist may contain duplicate .SUBCKT definitions where name and pin count match.

When you specify YES, if a duplicate subcircuit definition is detected that matches in name and in pin count, the tool chooses one of the .SUBCKT definitions and proceeds after issuing the warning. The .SUBCKT definition referred to in the warning is not used; rather, the named .SUBCKT definition not mentioned in any warning is the one that is used. The SPICE parser stops reporting warnings after 256 duplicate subcircuits are found.

If [LVS Spice Option S](#) is set, a warning is also generated if a duplicate subcircuit definition is detected that only matches in name, even if the pin count is different. Subcircuits having the same name but differing pin counts are considered to be different subcircuits by default. See “[Duplicate .SUBCKT Definitions](#)” in the *Calibre Verification User’s Manual*.

If you specify NO, then the warnings become fatal errors.

You may specify this statement once in your rule file.

### Examples

```
// Duplicate subcircuit definitions are allowed with warnings
LVS ALLOW DUPLICATE SUBCIRCUIT NAMES YES
```

Suppose the above statement is in your rule file, and the layout spice netlist has a duplicate .SUBCKT definition named “middle\_cell” at line 11. The following warning will appear in the run transcript:

```
LVS Netlist Compiler - Errors and Warnings for "layout.spi"
-----
Warning: Duplicate subckt definition "MIDDLE_CELL" at line 11 in file
"layout.spi"
```

## LVS Spice Allow Floating Pins

Specification statement

### LVS SPICE ALLOW FLOATING PINS {YES | NO}

#### Parameters

- **YES**

Keyword that instructs the tool to allow floating pins in subcircuit calls in SPICE netlists.  
This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs the tool not to allow floating pins in subcircuit calls in SPICE netlists.

#### Description

Specifies whether a SPICE subcircuit call may reference fewer pins than specified in the respective .SUBCKT definition.

You may specify this statement once in your rule file.

#### Examples

In the following example, subcircuit call X1 is valid by default. However, if you specify LVS Spice Allow Floating Pins NO in your rule file, the SPICE reader indicates an error for X1.

```
.SUBCKT ZZZ A B C  
...  
.ENDS  
  
X1 1 2 ZZZ      $ References only two out of three pins.
```

# LVS Spice Allow Inline Parameters

Specification statement

## LVS SPICE ALLOW INLINE PARAMETERS {NO | YES}

Used only in Calibre nmLVS/nmLVS-H and PERC.

### Parameters

- **NO**  
A required keyword that instructs the Calibre SPICE reader to disable support for inline parameter references for subcircuit calls in SPICE netlists, and to disable the default warning message issued when inline parameters are found in the SPICE netlist when this statement is not in the rule file.
- **YES**  
A required keyword that instructs the Calibre SPICE reader to enable support for inline parameter references for subcircuit calls in SPICE netlists.

### Description

Controls support for inline parameter references for subcircuit calls (X calls) in SPICE netlists. An inline parameter reference is a reference to a parameter that is defined on the same line within a subcircuit call. This statement is useful for matching the SPICE syntax interpretation of other tools used in your workflow that may be different from how Calibre interprets SPICE.

By default, the Calibre SPICE reader does not allow inline parameter references for subcircuit calls. Specifically, when parameter Y refers to parameter Z on a subcircuit instantiation line, it refers to the value of parameter Z in the instantiation environment (that is, the environment where the subcircuit call is resolved) and not to the value defined on the same subcircuit instantiation line.

For example, if the SPICE file contains the following line:

```
x1 node1 node2 subname a=1 b=a
```

the instantiation of subcircuit subname will have an environment where “a” has the value of 1 and “b” has the value of “a” in the calling environment.

In this code:

```
x0 A a=2
.subckt A
x1 node1 node2 subname a=1 b=a
...
.ends
```

parameter “b” has the value 2, which “b” gets from the “X0 A a=2” call. The definition a=1 is ignored in the assignment b=a when parameter “a” is referenced on the same line, whether or not “a” is otherwise defined.

If “a” is undefined in the calling environment, a warning is issued in the logfile similar to the following:

```
Warning: Undefined parameter "A" ignored in file "<file>" at line <number>
```

and “b” does not get a value.

When you specify YES, the Calibre SPICE reader uses parameter definitions when parsing parameter references on the same line, *as long as the definitions precede the references*. In the previous example, “a” is defined on the subcircuit call line to be 1, and this value is used when “a” is referenced in the definition of “b”. The inline parameter definition takes precedence over the definitions of the same parameter in the instantiation environment, if any exist.

Note that a premature inline parameter definition is not supported. For example, in the following line:

```
x1 node1 node2 subname b=a a=1
```

the subcircuit call assigns “a” with the value of 1 and “b” gets the value of “a” (if one is defined) in the instantiation of subcircuit subname, whether or not you specify YES.

If this statement is not specified in your rule file, Calibre issues the following warning message if an inline parameter definition is detected:

```
Parameter "<param>" references ambiguous parameters in file "<filename>"  
at line "<line number>", use LVS SPICE ALLOW INLINE PARAMETERS YES|NO to  
clarify.
```

This warning message can be disabled by explicitly specifying LVS Spice Allow Inline Parameters NO in your rule file.

Calibre allows inline parameter definitions for .SUBCKT definitions and this behavior is not affected by LVS Spice Allow Inline Parameters statement.

Notes:

- Different versions of SPICE-like simulators and other tools that import SPICE and similar netlist formats have different rules about inline parameters. The LVS Spice Allow Inline Parameters specification statement allows you to configure Calibre nmLVS to match better the interpretation of SPICE syntax used by other tools in the flow.
- Neither the source netlist nor the value of the parameter is affected by whether the value specified results in an undefined parameter error. That is, if an undefined parameter is reported and it could become defined if inline definitions were supported, then Calibre does not automatically turn on the support for inline SPICE parameters.
- Only subcircuit definitions, subcircuit calls, and .PARAM statements produce parameter values that can be referenced from other expressions. It is not possible in the SPICE language to reference, for example, the L parameter of a MOSFET device. Therefore, the change to SPICE parameter parsing and the LVS Spice Allow Inline Parameters statement does not affect the way parameters are evaluated for primitive devices such as MOSFETs or resistors.

## **Examples**

```
/* disables support for inline parameter references in SPICE netlists,  
and disables a warning message when an inline parameter reference is  
detected */  
LVS SPICE ALLOW INLINE PARAMETERS NO
```

## LVS Spice Allow Unquoted Strings

Specification statement

### LVS SPICE ALLOW UNQUOTED STRINGS {NO | YES}

#### Parameters

- **NO**

Keyword that instructs the tool to not allow unquoted string values in parameter assignments in SPICE netlists. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to allow unquoted string values in parameter assignments in SPICE netlists.

#### Description

Specifies whether SPICE parameter assignments can contain strings without double quotation marks ("").

If you specify YES, then a string parameter such as abc is permitted in a netlist, like this:

```
z1 = abc
```

An unquoted string parameter must begin with a letter or underscore (\_) character to qualify as valid when YES is specified.

If you specify NO (the default), then the previous example must appear in the netlist as this:

```
z1 = "abc"
```

If the string abc did not appear in quotes and NO was specified, the string would be treated as an undefined parameter name and a warning would be issued.

The NO setting is preferred because, when unquoted strings are allowed, references to undefined parameters are treated as literal strings. This causes the SPICE parser to be unable to generate warnings regarding the undefined parameters.

You may specify this statement once in your rule file.

#### Examples

```
// Do not allow unquoted string values for parameters.  
LVS SPICE ALLOW UNQUOTED STRINGS NO
```

# LVS Spice Conditional LDD

Specification statement

## LVS SPICE CONDITIONAL LDD {NO | YES}

### Parameters

- **NO**

Keyword that specifies that \$LDD designators in the netlist are processed unconditionally, whether a \*.LDD statement is present in the netlist, or not. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that specifies that \$LDD designators in the netlist are processed only if a \*.LDD comment-coded statement is also present in the netlist, and are ignored if there is no \*.LDD statement in the netlist.

### Description

Controls how the LVS SPICE reader processes \$LDD designators for M elements in SPICE netlists.

When you specify YES, \$LDD designators in the netlist are processed only if a \*.LDD comment-coded statement is also present in the netlist, and are ignored if there is no \*.LDD statement in the netlist. (The \*.LDD statement may appear anywhere in the netlist).

When you specify NO, \$LDD designators in the netlist are processed unconditionally, whether or not \*.LDD is present in the netlist.

You may specify this statement once in your rule file.

### Examples

```
// Process all $LDD designators unconditionally.
LVS SPICE CONDITIONAL LDD NO
```

# LVS Spice Cull Primitive Subcircuits

Specification statement

## LVS SPICE CULL PRIMITIVE SUBCIRCUITS {NO | YES}

### Parameters

- **NO**  
Keyword that specifies that all primitive subcircuits should be processed. This is the default.
- **YES**  
Keyword that specifies that instances of primitive subcircuits that do not appear in the rule file should be ignored and should not be processed by the LVS circuit comparison module.

### Description

Indicates whether the LVS SPICE parser should ignore primitive subcircuits having names that do not appear in the rule file. To qualify, names must appear either as element names in [Device](#) or [LVS Box](#) specification statements.

Checking of subcircuit names against LVS Box specification statements is subject to the SOURCE and LAYOUT keywords of the LVS Box statement. When reading a layout netlist (from [Layout Path](#)), the relevant LVS Box specification statements are those that apply to the layout; and when reading a source netlist (from [Source Path](#)), the relevant LVS Box specification statements are those that apply to the source. (Recall that, by default, an LVS Box specification statement applies to both layout and source).

**Case sensitivity** — When comparing subcircuit names against LVS Box specification statements, case sensitivity is controlled by the [Layout Case](#) setting for layout netlists and by the [Source Case](#) setting for source netlists. Comparison of subcircuit names against Device operations is always cases insensitive.

You may specify this statement once in your rule file.

## Examples

In this example, X1 and X2 are processed because the name AAA appears in a Device operation and the name BBB appears in a LVS Box statement. However, X3 is ignored because the name CCC does not appear in either.

Rule file:

```
DEVICE AAA SEED L(P)
LVS BOX BBB
LVS SPICE CULL PRIMITIVE SUBCIRCUITS YES
```

Netlist:

```
.SUBCKT AAA P
.ENDS

.SUBCKT BBB 1 2
.ENDS

.SUBCKT CCC 1 2
.ENDS

X1 1 AAA
X2 1 2 BBB
X3 1 2 CCC
```

## LVS Spice Implied MOS Area

Specification statement

### LVS SPICE IMPLIED MOS AREA {NO | YES}

#### Parameters

- **NO**

Keyword that specifies implied MOS areas are not calculated by LVS for M-type devices in SPICE netlists. This is the default behavior.

- **YES**

Keyword that specifies implied MOS areas are calculated by LVS for M-type devices in SPICE netlists.

#### Description

Indicates whether LVS should compute implied area values for MOS elements in SPICE source netlists (SPICE element M-type devices). When YES is specified, an area value is calculated using the formula:

$$A = W * L$$

The implied area value is available for tracing as property name A.

If W or L values are not specified, then A is not calculated. If an explicit “A” parameter is specified in the netlist, then it takes precedence, and the implied calculation is not used.

The following examples assume that LVS Spice Implied MOS Area YES is specified in the rule file:

```
M1 1 2 3 4 P W=3 L=2      $$ A = 3 * 2 = 6
M2 1 2 3 4 P W=3          $$ No implied A; L not specified.
M3 1 2 3 4 P W=3 L=2 A=8  $$ A = 8 is used.
```

You may specify this statement once in your rule file.

#### Examples

```
// Implied MOS area is not calculated.
LVS SPICE IMPLIED MOS AREA NO
```

# LVS Spice Multiplier Name

Specification statement

**LVS SPICE MULTIPLIER NAME** *name* [*name* ...]

## Parameters

- *name*

A required string that serves as a SPICE multiplier factor. Multiple *name* parameters can be specified.

## Description

Specifies parameter names that serve as multiplier factors in SPICE netlists instead of the standard SPICE parameter name M. One or more parameter names may be specified. All specified parameter names serve as multiplier factors wherever parameter M normally applies. This includes, but is not limited to, SPICE elements C, R, D, M, J, Q, L, V, as well as subcircuit calls X.

The name M itself may be specified also. If this statement is specified and it does not include the name M, then M does not serve as a multiplier factor and, moreover, its value is not available for tracing.

The values of multiplier factors are accessible for tracing (as in [Trace Property](#)) using property name M, regardless of the original parameter name used in the SPICE netlist. The original parameter name is not preserved.

Only one multiplier factor is allowed per SPICE element; duplicate multiplier factors per element are reported as errors even if different parameter names are used.

Generally, to serve as multiplier factor, a parameter must be specified in the SPICE netlist with explicit name in the form “name=value”; parameters with implied names (where only a value is specified in the netlist) do not serve as multiplier factors, even if the implied parameter name matches one of the multiplier names specified in this statement. The only exception is if the implied parameter name is M, then it may serve as a multiplier factor.

Parameter names in this statement are processed in a case-insensitive manner.

This statement may be specified any number of times and names are accumulated. If this statement is not specified, then the standard SPICE parameter name M is used.

## Examples

### Example 1

Basic syntax.

```
LVS SPICE MULTIPLIER NAME "MULT"
```

Only the name MULT is used as multiplier factor. For example:

```
C1 1 2 100 MULT=3      $$ Multiplier factor is 3.
C2 1 2 100 MULT=3 M=5  $$ Multiplier factor is 3.
C3 1 2 100 M=5        $$ No multiplier factor (defaults to 1).
```

### Example 2

Using more than one name.

```
LVS SPICE MULTIPLIER NAME "M" "AAA"
```

Either M or AAA may be used as multiplier factors. For example:

```
R1 1 2 100 M=4      $$ Multiplier factor is 4.  
R2 1 2 100 AAA=4   $$ Multiplier factor is 4.  
R3 1 2 100 M=4 AAA=4 $$ Error: duplicate multiplier factor.
```

### Example 3

Implied parameter names.

```
LVS SPICE MULTIPLIER NAME "C"
```

In the following line:

```
C1 1 2 100
```

the value 100 has implied parameter name C. Since the parameter name is implied, it does not serve as multiplier factor even though it appears in an LVS Spice Multiplier Name statement. This parameter is available for tracing as property C.

### Example 4

Tracing multiplier factors.

```
LVS SPICE MULTIPLIER NAME "MULT"  
TRACE PROPERTY MP M M 0
```

In the following line:

```
M1 1 2 3 4 P MULT=3
```

parameter MULT serves as multiplier factor and thus will be traced as property M by the Trace Property statement as shown.

# LVS Spice Option

Specification statement

## LVS SPICE OPTION [F] [S] [X]

### Parameters

At least one of the following must be specified in a space-delimited list:

- **F**  
Specifies the SPICE parser reports warnings for subcircuit calls with floating pins.
- **S**  
Specifies the SPICE parser reports warnings for subcircuit definitions with the same name, even if they have different numbers of pins.
- **X**  
Prevents the SPICE parser from modifying duplicate subcircuit call names. This option should be used with caution because when duplicate subcircuit call names are left unchanged, the SPICE parser may generate incorrect connectivity.

### Description

Specifies behavior of the SPICE parser.

When you use option F, a warning such as the following is issued for subcircuit (X) calls with floating pins:

```
Warning: Floating pins in file "n.float" at line 17 (4 pins)
```

This option is relevant only when floating pins are allowed ([LVS Spice Allow Floating Pins YES](#), the default). When floating pins are forbidden, they are reported as errors and option F is ignored.

When you use option S, a warning such as the following is issued for subcircuit definitions having the same name:

```
Warning: Duplicate subckt name "AAA" at line 9 in file "xx.spi"
```

Normally, the parser reports warnings for duplicate subcircuit definitions having the same name and the same number of pins. For example:

```
Warning: Duplicate subckt definition "AAA" at line 5 in file "xx.spi"
```

See [LVS Spice Allow Duplicate Subcircuit Names](#).

When you specify option S, subcircuit definitions having the same name but different numbers of pins also generate warnings. Such subcircuits are still classified as distinct, however. See “[Duplicate .SUBCKT Definitions](#)” in the *Calibre Verification User’s Manual*.

Normally, the SPICE parser appends strings of the form ==<number> to duplicate subcircuit call names to make them unique. Option X prevents this.

### Examples

```
//Report warnings for floating pins  
//Report warnings for all duplicate subcircuit names  
LVS SPICE OPTION F S
```

# LVS Spice Override Globals

Specification statement

## **LVS SPICE OVERRIDE GLOBALS {NO | YES}**

### Parameters

- **NO**

Specifies that global signals apply to all subcircuit pins throughout their internal hierarchies and cannot be overridden except through the [LVS Spice Prefer Pins](#) YES specification statement. This is the default behavior.

- **YES**

Specifies that an assignment of a signal to a global pin in a subcircuit call has the effect of overriding the respective global signal in the referenced subcircuit and in the entire sub-hierarchy below the subcircuit.

### Description

Specifies whether or not LVS allows pin assignments to override global signals in SPICE netlists. A global pin is any subcircuit pin with a global name (that is, a name that appears in a .GLOBAL statement in the netlist). This statement may appear once.

If you specify YES, then an assignment of a signal to a global pin in a subcircuit call has the effect of overriding the respective global signal in the referenced subcircuit and in the entire sub-hierarchy below the subcircuit. Overriding means that references to the global signal down the hierarchy of the subcircuit are treated, instead, as references to the signal that was assigned to the global pin in the subcircuit call. The override is per subcircuit call and applies only to the specific subcircuit call where the overriding assignment was made.

This statement operates both with regular positional pin assignment and with \$PINS-style pin assignment.

When you specify YES, LVS allows assignments to pins with global names in subcircuit calls even when those pins are not defined in the respective .SUBCKT statement. Such assignments may be entered with \$PINS constructs and have the same global-overriding effect described above. Also, .SUBCKT pins with global names refer to the respective global signals (as overridden) and are no longer distinct from the global signals themselves. As a result, the [LVS Spice Prefer Pins](#) specification statement has no effect in this circumstance.

When you specify NO (the default), or if this statement is omitted, then:

- Assignments to global pins in subcircuit calls have no effect unless LVS Spice Prefer Pins YES is specified. In the latter case, the pin assignment takes effect in the called subcircuit but not in the sub-hierarchy of the subcircuit.
- Assignments to undefined pins are not allowed in subcircuit calls.
- .SUBCKT pins with global names are distinct from global signals with the same names. Resolution of node names within such subcircuits is controlled with the [LVS Spice Prefer Pins](#) specification statement.

**Global pins in hcells** — Hierarchical LVS adds global pins (ports) to hcells, if necessary, to pass global signals through the hierarchy. This process is maintained whether or not [LVS Spice Override Globals](#) is requested. Pins that are added to hcells in this manner receive their names from the global signals that trigger their creation, not from any overriding signals, if present. See the [Hcell](#) statement for information about hcell declarations.

**LVS netlists** — When LVS Spice Override Globals YES is specified, netlists created with the LVS Write {[Layout](#) | [Source](#)} Netlist specification statements have all global signals explicitly routed through subcircuit pins. Normally, global signals are represented with .GLOBAL statements in those netlists.

## Examples

### Example 1

Consider the following SPICE netlist excerpt:

```
.GLOBAL VCC

.SUBCKT SUB1 A B
M1 A B VCC VCC P
.ENDS

.SUBCKT SUB2 C D
M2 C D VCC VCC P
X3 C D SUB1
.ENDS

.SUBCKT TOP
X4 SUB2 $PINS C=E D=F VCC=AVCC
X5 SUB2 $PINS C=E D=F VCC=DVCC
X6 SUB2 $PINS C=E D=F
.ENDS
```

Node AVCC is assigned to pin VCC in X4 in TOP. Normally, this assignment would not be allowed because the respective subcircuit SUB2 does not have a pin called VCC. With LVS Spice Override Globals YES, the assignment is allowed. As a result, any reference to VCC within the sub-hierarchy of X4 is interpreted as a reference to AVCC. For example, devices X4/M2 and X4/X3/M1 are now both connected to AVCC; normally, they would be connected to global VCC.

In X5 in TOP, node DVCC is assigned to pin VCC. As a result, devices X5/M2 and X5/X3/M1 are both connected to DVCC.

In X6 in TOP, there is no assignment to VCC. As a result, global VCC is used down the hierarchy of X6. Devices X6/M2 and X6/X3/M1 are both connected to global VCC.

If SUB1 and SUB2 were used as hcells in hierarchical LVS, then they would each receive a VCC port. The VCC port of X4 would be connected to AVCC in TOP, the VCC port of X5 would be connected to DVCC in TOP, and the VCC port of X6 would be connected to net VCC in TOP. The VCC port of X3 would be connected to net VCC in SUB2.

**Example 2**

Consider the following SPICE netlist excerpt:

```
.GLOBAL VCC

.SUBCKT SUB1
C1 1 VCC
.ENDS

.SUBCKT SUB2 VCC
X2 SUB1
C3 2 VCC
.ENDS

.SUBCKT TOP
X4 ABC SUB2
.ENDS
```

Node ABC is passed to an existing pin VCC of SUB2 (this time using regular positional pin assignment). Normally, devices X4/C3 and X4/X2/C1 would be connected to global VCC, not to ABC. With LVS Spice Prefer Pins YES, device X4/C3 would be connected to ABC, but device X4/X2/C1 would be connected to global VCC. With LVS Spice Override Globals YES, both devices X4/C3 and X4/X2/C1 are connected to ABC.

# LVS Spice Prefer Pins

Specification statement

## LVS SPICE PREFER PINS {NO | YES}

### Parameters

- **NO**

Keyword that instructs the tool to prefer global nets over subcircuit pins when resolving net names in SPICE netlists. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to prefer subcircuit pins over global nets when resolving net names in SPICE netlists.

### Description

Specifies whether to prefer subcircuit pins over global nets when resolving net names in SPICE netlists. This statement may be specified once in your rule file.

### Examples

```
.GLOBAL VCC VSS  
  
.SUBCKT MYCELL VCC VSS  
C1 VCC VSS  
.ENDS  
  
X1 A B MYCELL
```

Normally, capacitor C1 is connected to the global nets VCC and VSS respectively. However if LVS Spice Prefer Pins YES is specified, then C1 is connected to the VCC and VSS pins of MYCELL. These are in turn connected to top level nets A and B.

# LVS Spice Redefine Param

Specification statement

## LVS SPICE REDEFINE PARAM {NO | YES}

### Parameters

- **NO**

Keyword that instructs the tool to disallow .PARAM value redefinition. All .PARAM statements apply globally. This is the default behavior if you do not specify this statement.

- **YES**

Keyword that instructs the tool to allow .PARAM value redefinition. All .PARAM statements that appear within subcircuit definitions are used locally within that subcircuit only.

### Description

Specifies whether to allow .PARAM values to be redefined throughout a SPICE netlist.

If you specify NO is specified, then .PARAM redefinition is not supported; for each parameter name, the first .PARAM value specified for the parameter in the netlist is used and all other values are ignored with warnings given. All .PARAM statements then apply globally, regardless of whether they appear inside or outside of subcircuits.

If you specify YES, then .PARAM redefinition is supported; for a given parameter name, at any point in the netlist, the latest .PARAM value specified for the parameter is used. Latest is defined statically and does not depend on the order of subcircuit calls. The .PARAM statements that appear within subcircuit definitions are used locally in those subcircuits only. The .PARAM statements that appear outside of subcircuit definitions (the usual case) apply globally.

Numeric-valued .PARAM parameters may not be redefined as string-valued by a subsequent .PARAM statement, and vice-versa.

## Examples

```
.PARAM AA=1

.SUBCKT SSA
C1 1 2 C=AA
.ENDS

.PARAM AA=2

.SUBCKT SSB
C2 1 2 C=AA
.PARAM AA=3
C3 1 2 C=AA
.ENDS

.SUBCKT TOP
C4 1 2 C=AA
X5 SSA
X6 SSB
.PARAM AA=4
X7 SSA
X8 SSB
C9 1 2 C=AA
.ENDS
```

In this netlist, if LVS Spice Redefine Param YES is specified, then:

- C1 has value C=1 in all instances of SSA, both X5 and X7.
- C2 has value C=2 in all instances of SSB, both X6 and X8.
- C3 has value C=3 in all instances of SSB, both X6 and X8.
- C4 has value C=2 from the latest global-scope .PARAM statement, AA=2.

Note that the .PARAM AA=3 definition inside of SSB has local scope and thus does not apply to C4.

- C9 has value C=4.

If LVS Spice Redefine Param NO is specified, then .PARAM AA=1 applies everywhere, all capacitors have values C=1 and warnings are issued for the redefined .PARAM values.

# LVS Spice Rename Parameter

Specification statement

**LVS SPICE RENAME PARAMETER** *old\_param new\_param*

```
{ [C] [D] [J] [L] [M] [Q] [R] [ ( component_subtype ) ] }
{ [SUBCKT] [X] [ ( subcircuit_name ) ] }
[V] [PARAM] [OPTIONS]
[LEFT] [RIGHT]
[SOURCE] [LAYOUT]
```

Used only in Calibre nmLVS/nmLVS-H and PERC.

## Summary

This statement causes SPICE parameters to be renamed from an existing name to a different name during LVS comparison in all places where the existing name appears in the source and layout netlists. This is the default behavior. You can optionally select the SPICE elements in which the renaming occurs.

## Parameters

- ***old\_param***  
A required SPICE parameter that is to be renamed. This parameter appears in the source or layout netlists.
- ***new\_param***  
A required SPICE parameter name that is substituted in place of ***old\_param*** during LVS comparison. This parameter appears in the source or layout netlists.
- C  
An optional keyword that specifies the parameter renaming occurs for capacitor elements.
- D  
An optional keyword that specifies the parameter renaming occurs for diode elements.
- J  
An optional keyword that specifies the parameter renaming occurs for JFET elements.
- L  
An optional keyword that specifies the parameter renaming occurs for inductor elements.
- M  
An optional keyword that specifies the parameter renaming occurs for MOSFET elements.
- Q  
An optional keyword that specifies the parameter renaming occurs for bipolar transistor (BJT) elements.

- R  
An optional keyword that specifies the parameter renaming occurs for resistor elements.
- *(component\_subtype)*  
An optional component subtype (or model name) enclosed in parentheses, to which the parameter renaming applies. The *component\_subtype* may only be specified with *one* of the keywords C, D, J, M, L, Q, or R in a single LVS Spice Rename Parameter statement.
- SUBCKT  
An optional keyword that specifies the parameter renaming occurs for .SUBCKT definitions.
- X  
An optional keyword that specifies the parameter renaming occurs for subcircuit calls.
- *(subcircuit\_name)*  
An optional subcircuit name enclosed in parentheses, to which the parameter renaming applies. The *subcircuit\_name* may only be specified with either the X or SUBCKT keywords, but not both, in a single LVS Spice Rename Parameter statement.
- OPTIONS  
An optional keyword that specifies the parameter renaming occurs in .OPTIONS statements.
- PARAM  
An optional keyword that specifies the parameter renaming occurs in .PARAM statements.
- V  
An optional keyword that specifies the parameter renaming occurs for voltage source elements.
- LEFT RIGHT | LEFT | RIGHT  
Optional keywords that specify which side of an equals sign (=) the parameter renaming occurs. The choices are as follows:
  - LEFT RIGHT — Specifies that parameters should be renamed on both sides of an equals sign. This is the default behavior.
  - LEFT — Specifies that the parameters should be renamed only on the left side of an equals sign.
  - RIGHT — Specifies that the parameters should be renamed only on the right side of an equals sign.
- SOURCE  
An optional keyword that specifies the statement applies to the source design. This option is used by default.

- **LAYOUT**

An optional keyword that specifies the statement applies to the layout design. This option is used by default.

### Description

Specifies that the SPICE parameter indicated by *old\_param* be replaced with *new\_param* as SPICE netlists are read during LVS comparison. This statement can be used to accommodate SPICE netlists that come from third-party sources and that have different parameter naming conventions from Calibre. Such netlists often require the renaming of parameters to process the SPICE netlist.

The C, D, J, L, M, Q, R, V, X, SUBCKT, PARAM, and OPTIONS keywords are all optional. This set of keywords is referred to as *statement-type specifiers*. They correspond to SPICE elements and control statements of the same names. Any set of these keywords may be specified in an LVS Spice Rename Parameter statement, so long as no *component\_subtype* (also known as model name) or *subcircuit\_name* parameters are specified. If a *component\_subtype* or *subcircuit\_name* is specified, then only one statement-type specifier is allowed in that LVS Spice Rename Parameter statement. By default none of these keywords are explicitly specified, and the parameter renaming applies to all of these SPICE elements and control statements in the netlists.

The optional *component\_subtype*, which is specified in parentheses, must be used with only one of the C, D, J, L, M, Q, or R statement-type specifiers, and this restricts the renaming to statements of the specified element type and subtype.

The optional *subcircuit\_name*, which is specified in parentheses, must be used with only one of the X or SUBCKT statement-type specifiers, and this restricts the renaming to statements of the specified subcircuit name.

The V, PARAM, and OPTIONS do not have subtypes or subcircuit names.

Again, the default is to rename parameters on all of the above statement types if no statement-type specifier is used.

The optional LEFT and RIGHT keywords determine whether the replacement takes place on the LEFT or RIGHT side of all “=” (equal signs) on the relevant lines.

All specification statements that reference *old\_param* property names, such as [Trace Property](#), [LVS Reduce](#), [LVS Split Gate Ratio](#), [LVS Filter](#), and so forth, should reference the *new\_param* name. This applies property names in various property and effective property computation languages, and care should be taken that the correct property names are used in such programs when using the LVS Spice Rename Parameter statement.

The renaming takes place at the lowest level of the parameter parsing system, so all other functionality that references parameter names or values sees the new name. For example, if the netlist contains the following line:

```
R1 A B 5K M=10 N=20
```

and the rule file contains the following specification statements:

```
LVS SPICE RENAME PARAMETER M Z  
LVS SPICE RENAME PARAMETER N M  
LVS SPICE REPLICATE DEVICES YES
```

then the device is replicated 20 times, not 10, because all operations in Calibre treat this line as if it were this:

```
R1 A B 5K Z=10 M=20
```

Note that the LEFT and RIGHT keywords allow manipulation of statements such as this:

```
X1 A B SUBCKT A=5 B=10 C=A+2
```

For example, to change the reference to “A” in the definition of “C” to reference “B” instead, you can rename parameters as follows:

```
LVS SPICE RENAME PARAMETER A B X RIGHT
```

and the line is interpreted as if it read:

```
X1 A B SUBCKT A=5 B=10 C=B+2
```

To swap the values of “A” and “B”, while having “C” still reference “A”, you can rename parameters as follows:

```
LVS SPICE RENAME PARAMETER A B X LEFT  
LVS SPICE RENAME PARAMETER B A X LEFT
```

and the resulting line becomes:

```
X1 A B SUBCKT B=5 A=10 C=A+2
```

Note that parameter name matching for the renaming is subject to the case-sensitivity rules applicable to each type of the parameter. For example, all parameters of primitive devices are matched in a case-insensitive manner. On the other hand, case sensitivity of subcircuit parameters is controlled by the case sensitivity settings for the relevant netlist (SOURCE or LAYOUT).

Notes:

- The LVS Spice Rename Parameter specification statement allows parameters to be renamed but cannot be used to assign values to them. Specifically, neither *old\_param* nor *new\_param* names can be a number.
- Renaming built-in W and L properties of MOS devices has additional limitations. For example, by default, LVS interprets any property whose name begins with “w” or “W” as width. Renaming one “width” property to a different “width” property has no effect. The following statement:

```
LVS SPICE RENAME PARAMETER W WIDTH M
```

is valid but has no effect because WIDTH and W properties are treated the same way. Similar limitations exist for “l”, “L”, and “length”.

This behavior can be disabled with the [LVS Spice Strict WL](#) statement, in which case there are no additional restrictions on built-in properties of MOS devices. In any case, built-in properties can be renamed to non-built-in properties, or even to other built-in properties. For example, the following two statements swap L and W on all MOS devices:

```
LVS SPICE RENAME PARAMETER W L M
LVS SPICE RENAME PARAMETER L W M
```

- If more than one LVS Spice Rename Parameter statement applies to the same property of the same SPICE element, a compilation error is reported. For example, it is not possible to override a less-specific statement with a more-specific one:

```
LVS SPICE RENAME PARAMETER R CAT R(NWRES)
LVS SPICE RENAME PARAMETER R DOG R // error, duplicate element
```

These statements do not cause all R properties on resistors to be renamed to DOG except for resistors with model name NWRES. Instead, they are considered ambiguous and an error results. To rename parameters of the same element differently, based on model name, you must explicitly list all necessary model names. For example:

```
LVS SPICE RENAME PARAMETER R CAT R(NWRES)
LVS SPICE RENAME PARAMETER R DOG R(PRES)
LVS SPICE RENAME PARAMETER R DOG R(NDPRES)
LVS SPICE RENAME PARAMETER R DOG R(PDRES)
```

- LVS Spice Rename Parameter statements should not be used to rename layout parameters when connectivity extraction (that is, the Layout System is not SPICE) and netlist comparison are done in a single nmLVS-H run. Instead, parameter names should be changed in [Device](#) definitions themselves. The reason for this is, when netlist extraction and comparison are done in a single step, the Layout System is geometric. This triggers certain consistency-checking algorithms that ensure extracted DEvice parameters will match what the comparison stage of LVS expects to see. If extracted DEvice parameters are being renamed by an LVS Spice Rename Parameter statement in such a case, this can cause an error due to the consistency checking algorithms.
- If built-in parameters of primitive devices are renamed, they are no longer subject to the special treatment for built-in properties. For example, default property reduction programs do not apply, partner properties are not automatically made available, .OPTIONS SCALE has no effect on the renamed properties, and so forth. However, if a built-in property is renamed to a different built-in property, then the special rules for the new built-in property do apply.
- It is not possible to rename the unnamed, or positional, SPICE parameters. For example, the statement:

```
LVS SPICE RENAME PARAMETER C CVAL C
```

has no effect on the unnamed C property:

```
C1 1 2 100
```

- When ambiguous parameter renaming is detected, the following error message is issued when the netlist is parsed:

```
Error: Parameter "<name>" ambiguously renamed in file "<netlist>" at  
line <line>
```

However, for MOS devices with three pins, a SPICE syntax error is reported instead.

- If \*.EQUIV statements are used on model names, then LVS Spice Rename Parameter statements should use the model names as written in the SPICE netlist, before \*.EQUIV is applied.

See also [LVS Spice Redefine Param.](#)

## Examples

This example shows how to rename a parameter “area” to “a” for all occurrences in layout and source, and on the left side of an equals sign:

```
LVS SPICE RENAME PARAMETER area a LEFT  
  
TRACE PROPERTY C a a // ensure all references to property "a" are correct
```

# LVS Spice Replicate Devices

Specification statement

## LVS SPICE REPLICATE DEVICES {NO | YES}

### Parameters

- **NO**

Specifies that the M property in SPICE netlists does not cause physical copies of elements owning M parameters to be made in the SPICE netlist. This is the default if you do not include this statement in a rule file.

- **YES**

Specifies that the M property in SPICE netlists causes creation of multiple physical copies of elements owning M parameters in the SPICE netlist.

### Description

Specifies whether the LVS SPICE parser should physically replicate device elements in a SPICE netlist that own the M parameter. This specification statement may appear in the rule file once.

In SPICE netlists, property M is evaluated immediately and not handed down through parameter passing. If you specify YES, for every SPICE device element that owns an M parameter, the parser creates M physical copies of the device element, connected in parallel. All copies are identical except for element names. The first copy receives the element name specified in the netlist. Subsequent copies receive that element name followed by character strings of the form ==2, ==3, ... ==n respectively, where n is the serial number of the copy. For example, R10, R10==2, R10==3, and so forth.

All copies own the M property with value 1 (not the original M value). All other parameter values are original values as specified in the netlist; the M parameter in this mode does not modify other parameter values. M parameters in this mode are required to have positive integer values within integer range (specifically, unsigned long integer: 1 to  $2^{32}-1$  in 32-bit mode and 1 to  $2^{64}-1$  in 64-bit mode). If M=1 then only one copy of the device element is created.

If you specify NO, or if this statement is omitted from the rule file, then the M parameter in SPICE netlists does not create multiple physical copies of the owner element. Instead, a single copy of the element is created and the M parameter merely modifies the values of certain other parameters of the element, as appropriate (see the “[SPICE Format](#)” Chapter in the *Calibre Verification User’s Manual*). The element owns the M property with the original M value as indicated in the netlist. M parameters in this mode are not limited to positive integer values and may have fractional and/or negative values.

This specification statement operates on SPICE device elements, including R, C, L, D, Q, J, M, V and other device elements. Subcircuit calls (X elements) are not affected by this statement; the M parameter in subcircuit calls always creates multiple physical copies of the subcircuit.

Note that a YES setting may carry a penalty in performance and process size, because more device instances must be created and processed. If you specify a YES value and also disable

parallel device reduction then the performance penalty may be more severe because you introduce ambiguity to the design.

### Examples

#### Example 1

Netlist:

```
C1 1 2 C=10 AAA=20 M=3
```

If you specify YES, then the SPICE parser creates three physical copies of the capacitor, connected in parallel, each copy having the original values for C and AAA and an M value of 1:

Element Name	Pins	Properties
C1	pos=1, neg=2	c=10, aaa=20, m=1
C1==2	pos=1, neg=2	c=10, aaa=20, m=1
C1==3	pos=1, neg=2	c=10, aaa=20, m=1

If you specify NO, or if this statement is omitted, then the SPICE parser creates a single copy of the capacitor, with a multiplied value for C, original value for AAA, and original value for M:

Element Name	Pins	Properties
C1	pos=1, neg=2	c=30, aaa=20, m=3

#### Example 2

If you specify YES, LVS can match a group of parallel devices to a single SPICE device that owns an M parameter, even if parallel device reduction is disabled. For example:

Rule file:

```
LVS SPICE REPLICATE DEVICES YES
LVS REDUCE PARALLEL CAPACITORS NO
TRACE PROPERTY C C C 0
TRACE PROPERTY C AAA AAA 0
```

Layout:

```
C1 1 2 C=10 AAA=20
C2 1 2 C=10 AAA=20
C3 1 2 C=10 AAA=20
```

Source:

```
C1 1 2 C=10 AAA=20 M=3
```

With these settings, the layout and source match. Notice, however, that if you specify NO, or if you omit this statement, then LVS tells you that there are three capacitor devices in the layout but only one in the source.

**Example 3**

In this example we see how to use LVS Spice Replicate Devices YES to apply effectively the M parameter in SPICE to properties that normally are not affected by that parameter, for example, user-defined properties.

Rule file:

```
LVS SPICE REPLICATE DEVICES YES
LVS REDUCE PARALLEL CAPACITORS YES [
  EFFECTIVE AAA
  AAA = SUM(AAA)
]
TRACE PROPERTY C AAA AAA 0
```

Layout:

```
C1 1 2 AAA=20
C2 1 2 AAA=20
C3 1 2 AAA=20
```

Source:

```
C1 1 2 AAA=20 M=3
```

With these settings, the layout and source initially contain three capacitors each. The capacitors are reduced in parallel and the effective property value for AAA is 60 in both layout and source. Notice that if you specify LVS Spice Replicate Devices NO, or if you omit this statement, then LVS reports a discrepancy for property AAA, because AAA is 60 in the layout but 20 in the source.

## LVS Spice Scale X Parameters

Specification statement

### LVS SPICE SCALE X PARAMETERS {NO | YES | MOS}

#### Parameters

- **NO**

Keyword that instructs the tool not to apply .OPTIONS SCALE factors to built-in devices that are instantiated as X elements. This is the default behavior.

- **YES**

Keyword that instructs the tool to apply .OPTIONS SCALE factors to all built-in devices that are instantiated as X elements.

- **MOS**

Keyword that instructs the tool to apply .OPTIONS SCALE factors to MOSFET built-in devices that are instantiated as X elements. No other devices are affected.

#### Description

Specifies whether .OPTIONS SCALE factors in SPICE netlists apply to built-in devices instantiated as X elements. This applies to LVS comparison. By default, scale factors are not applied.

For a complete discussion of how .OPTIONS SCALE factors are applied (either YES or MOS is specified), see “[X Instantiated Devices](#)” and “[X Instantiated MOSFET Devices](#)” in the *Calibre Verification User’s Manual*.

This statement was released in 2009.1. It is also available in 2008.3 UR2 and later 2008.3 releases, and 2008.4 UR1 and later 2008.4 releases. It is not available in the 2008.4 quarterly release. [Table 4-28](#) summarizes the history of default behaviors.

**Table 4-28. Scaled X Device Default Behavior History**

Release	Default Behavior
2008.1 and earlier	Equivalent to NO
2008.2	Equivalent to MOS
2008.3, 2008.4	Equivalent to YES
2009.1 and later	NO

## Examples

Consider the following SPICE source netlist:

```
.options scale=1e-06

.subckt mp d g s
.ends

.subckt r pos neg
.ends

x0 a b c mp l=3 w=10 $[p]
m1 d e f p l=5 w=8

x1 g h r r=1k l=.1 w=.2 $[r]
r1 j h r r=1k l=.1 w=.2
```

and the following layout netlist:

```
m0 1 2 3 p l=3e-06 w=10e-06
m1 4 5 6 p l=5e-06 w=8e-06

r0 7 8 r l=1e-07 w=2e-07 r=1000
r1 9 8 r l=1e-07 w=2e-07 r=1000
```

The instances x0 and x1 are MOSFET and resistor devices, respectively. Because they were netlisted as X elements in the source, the scaling factor is not applied by default, and the devices will not match because the width parameters differ. Therefore, with LVS Spice Scale X Parameters NO, the result will be INCORRECT, with property comparison errors for the width properties of both x0 and x1.

If MOS is specified, instance x0 will be scaled and correctly matched, but the comparison result will still be INCORRECT because instance x1 will not be scaled.

If YES is specified, both x0 and x1 will be scaled and the comparison result will be CORRECT.

## LVS Spice Slash Is Space

Specification statement

### LVS SPICE SLASH IS SPACE {YES | NO}

#### Parameters

- **YES**

Keyword that instructs the tool to treat the slash character (/) as white space within subcircuit definitions and subcircuit calls in SPICE netlists. This is the default behavior if you do not include this statement in the rule file.

- **NO**

Keyword that instructs the tool not to treat the slash character (/) as white space within subcircuit definitions and subcircuit calls in SPICE netlists.

#### Description

Specifies whether the slash character (/) should be treated as white space within subcircuit definitions and subcircuit calls in SPICE netlists.

You can include slashes as literal characters within pin or node names, but this is not recommended because the slash character serves as hierarchical delimiter in LVS.

You can specify this statement once in your rule file.

#### Examples

##### Example 1

In the following example, assume the following subcircuit definition:

```
.SUBCKT SSS A B/C  
...  
.ENDS
```

The default output is three pins (A, B, C). However, if you specify LVS Spice Slash Is Space NO in your rule file, the output is two pins (A, B/C).

##### Example 2

In the following example, assume the subcircuit call:

```
X1 1 2/3 SSS
```

The default output is three nets (1, 2, 3). However, if you specify LVS Spice Slash Is Space NO in your rule file, the output is two nets (1, 2/3).

# LVS Spice Strict WL

Specification statement

## LVS SPICE STRICT WL {NO | YES | NONE}

### Parameters

- **NO**

Keyword that instructs LVS to interpret any SPICE netlist parameter beginning with a W as width and beginning with an L as length. This applies to M devices. This is the default behavior if you do not specify this statement.

- **YES**

Keyword that instructs LVS to interpret only the letter “W” for width and only the letter “L” for length in SPICE netlists. This applies to M devices.

- **NONE**

Keyword that instructs LVS to interpret any SPICE netlist parameter beginning with a W as width and beginning with an L as length. This applies to M devices and to X devices instantiated as MOSFETs.

### Description

Controls the parsing of width and length parameters for MOSFET elements in SPICE netlists.

If NO is specified, then any parameter name that begins with the letter W is interpreted as width and any parameter name that begins with the letter L is interpreted as length (these letters may be upper- or lowercase). This applies to SPICE element M.

If you have more than one parameter that begins with either the letter W or the letter L for an M element, then specifying NO (or accepting it as the default) causes this situation to be an error because there cannot be more than one property that specifies either width or length.

If YES is specified, then only parameters with the exact names “W” and “L” (upper- or lowercase) are interpreted as width and length, respectively. This allows you to have additional properties that begin with letters W or L.

If NONE is specified, then the same behavior as the NO keyword is enforced for M elements. In addition, the same behavior applies to X elements that are instantiated as MOSFETs. See “[X Instantiated MOSFET Devices](#)” in the *Calibre Verification User’s Manual*.

## Examples

Consider the following two SPICE instantiations:

```
mp1 a b c d MP width=2 l=3  
xp1 a b c d MP width=2 l=3
```

Assuming a SCALE factor of 0.3e-6, the following would be the three interpretations for the keywords of LVS Spice Strict WL:

NO—

```
mp1 a b c d P w=.6e-6 l=.9e-6 $ width becomes w for M elements  
xp1 a b c d MP width=2 l=.9e-6 $ width remains width for X elements
```

NONE—

```
mp1 a b c d P w=.6e-6 l=.9e-6 $ width becomes w  
xp1 a b c d MP w=.6e-6 l=.9e-6 $ in both cases
```

YES—

```
mp1 a b c d P width=2 l=.9e-6 $ width remains width  
xp1 a b c d MP width=2 l=.9e-6 $ in both cases
```

The SCALE factor is applied correctly for all parameters. The parameter “l” remains “l” in all cases.

# LVS Split Gate Ratio

Specification statement

**LVS SPLIT GATE RATIO** *component\_type property\_name* [‘(‘*spice\_parameter*‘)’]

*tolerance*

[SOURCE] [LAYOUT]

[DIRECT] [MASK]

## Summary

This statement provides instructions for split-gate ratio property checking in LVS. It is used when **LVS Reduce Split Gates YES** is specified.

## Parameters

- *component\_type*

A required **Device** component type to which this statement applies. LVS Split Gate Ratio applies only to split gate structures; therefore, MOS transistor types and equivalent types specified in an **LVS Device Type** statement apply. Other component types can be specified, but the only effect is that LVS attempts to read the respective properties from the database.

- *property\_name*

A required name of a property to be checked. It applies only to instances that own the specified property.

- (‘*spice\_parameter*’)

An optional string, which must be in parentheses, that is an uppercase or lowercase letter specified with the *property\_name* parameter. The letter tells LVS to treat property values as strings in SPICE-like syntax (see the section “Spice-Like Property Syntax” in the *ICverify User’s and Reference Manual*) and to parse the strings to obtain the specified SPICE value. Possible letters are as follows:

**W** MOS width

**C** capacitance

**L** MOS length

**A** diode area

**R** resistance

**P** diode perimeter

When this parameter is present, the check applies only to instances that own a property with the specified *property\_name* parameter. The property value contains the specified *spice\_parameter*.

- *tolerance*

A required non-negative, floating-point number, which specifies the tolerance in percent for reporting discrepancies. It may be a numeric variable.

- SOURCE

An optional keyword that specifies the statement applies to the source design. This option is used by default.

- **LAYOUT**

An optional keyword that specifies the statement applies to the layout design. This option is used by default.

- **DIRECT**

An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.

- **MASK**

An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

### Description

Specifies to check property ratios in split gate structures.

It computes column-to-column ratios of the specified property for devices in each row of the split gate structure and then compares the ratios computed in different rows. (Note the distinction between row and column is arbitrary.) The smallest ratio serves as base and other ratios are checked against it. Discrepancies are reported when the difference exceeds the specified tolerance. This statement applies only when split gate reduction is performed ([LVS Reduce Split Gates YES](#)).

In split gates that have more than two columns, one column serves as reference and the others are compared to it independently.

Only MOS transistor types are relevant. Other types can be specified, but the only effect is that LVS attempts to read the respective properties from the database.

Comparisons of *component\_type* are case-sensitive if the [LVS Compare Case](#) specification statement is specified with the YES or TYPES optional keywords.

This statement may be specified multiple times. A (*component\_type*, *property\_name*) pair can only be specified once per DIRECT SOURCE, DIRECT LAYOUT, MASK SOURCE, MASK LAYOUT combination.

Usage is as follows:

- In Calibre nmLVS, the MASK SOURCE set of LVS Split Gate Ratio specification statements applies to the [Source System](#) and the MASK LAYOUT set applies to the [Layout System](#). Layout devices may be extracted.
- In ICtrace DIRECT mode, the DIRECT SOURCE set of LVS Split Gate Ratio specification statements applies to the schematic and the DIRECT LAYOUT set applies to the layout. Layout devices are not extracted.
- In ICtrace MASK mode, the MASK SOURCE set of LVS Split Gate Ratio specification statements applies to the schematic and the MASK LAYOUT set applies to the layout. Layout devices are not extracted.

Any [Device](#) statements having element names that match a MASK mode LVS Split Gate Ratio statement component type will have their split gate ratios computed by the corresponding LVS Split Gate Ratio statement setting. Consistency checking against the rule file Device statements is performed as follows:

- A (*spice\_parameter*) may not be specified as part of the *property\_name* being checked. This is because properties for extracted devices do not have SPICE values.
- There must be at least one Device definition corresponding to the LVS Split Gate Ratio specification statement.
- The *property\_name* must be computed in all Device definitions that correspond to the LVS Split Gate Ratio specification statement.
- The *property\_name* must be computed with correct type (numeric) in all Device definitions that correspond to the LVS Split Gate Ratio specification statement.
- The *property\_name* is not a vector type in any Device operation that corresponds to the LVS Split Gate Ratio statement.

The following rules govern which LVS Split Gate Ratio specification statements undergo consistency checking:

- In Calibre nmLVS, if the [Source System](#) is a layout, consistency checking is performed on all LVS Split Gate Ratio specification statements in the MASK SOURCE set.
- In Calibre nmLVS, if the [Layout System](#) is a layout, consistency checking is performed on all LVS Split Gate Ratio specification statements in the MASK LAYOUT set.
- In ICtrace, consistency checking is performed on all LVS Split Gate Ratio specification statements in the MASK LAYOUT set.

Discrepancies are reported in the LAYOUT ERRORS section of the LVS Report for layout devices and in the SOURCE ERRORS section for source devices. The report lists individual devices in each row along with respective property values, computed ratio and error percentage. The base row and all error rows are indicated; correct rows are not listed. Following is an example of the error report:

```
1 property w
base: m1 MP(P): 1u, m4 MP(P): 4u. ratio: 4
m2 MP(P): 2u, m5 MP(P): 9u. ratio: 4.5 error:12.5%
m3 MP(P): 3u, m6 MP(P): 15u. ratio: 5 error: 25%
```

## Examples

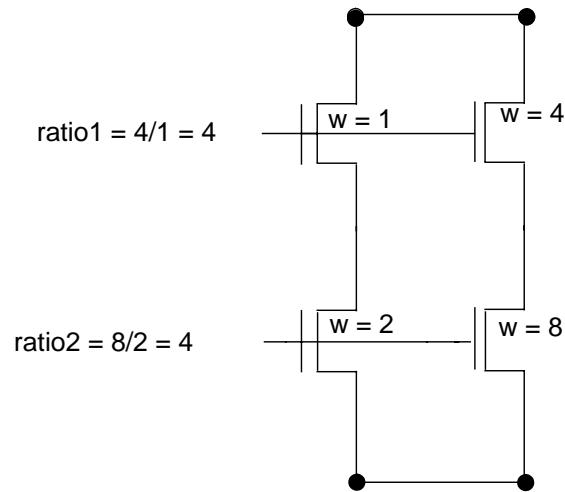
In the following examples, assume the following statement:

```
LVS SPLIT GATE RATIO MP W 10
```

### Example 1

In Figure 4-166, the ratios between W values are equal in both rows; there is no discrepancy.

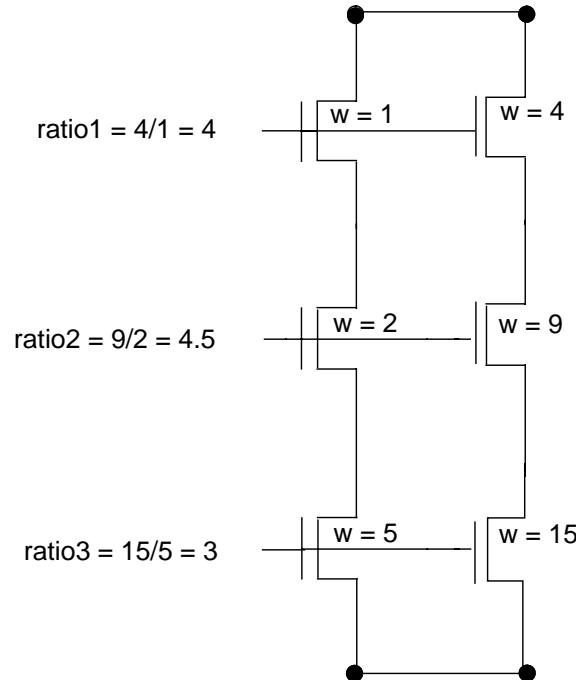
**Figure 4-166. LVS Split Gate Ratio, Example 1**



**Example 2**

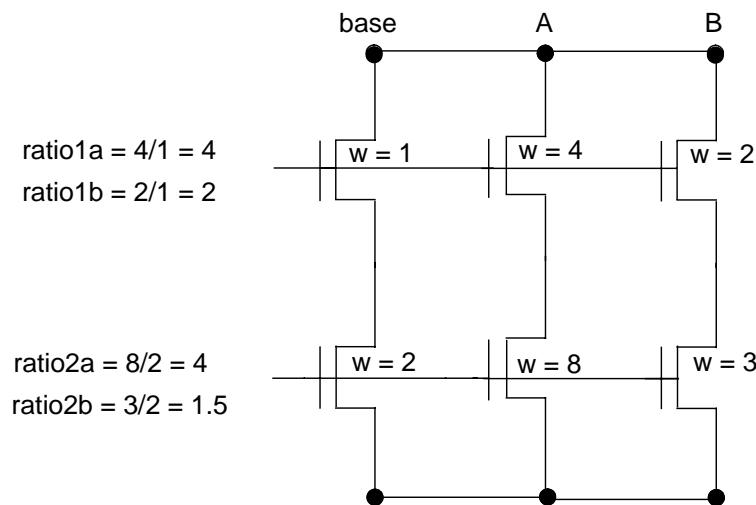
In Figure 4-167, the ratio between W values in the first row is 4, the ratio in the second row is 4.5, and the ratio in the third row is 5. The smallest ratio is 4 in the first row, and it serves as base. The difference between ratios in the second and first rows is  $(4.5-4)/4 = 12.5\%$ . The difference between ratios in the third and first rows is  $(5-4)/4 = 25\%$ . Both differences are larger than the specified tolerance of 10%, and discrepancies are reported in both rows.

**Figure 4-167. LVS Split Gate Ratio, Example 2**



**Example 3**

In Figure 4-168, the left column was chosen as base. Comparing column A against base, the ratios are 4 in both rows and there is no discrepancy. However, comparing column B against base, the ratio in the first row is 2 and the ratio in the second row is 1.5. The difference is  $(2-1.5)/1.5 = 33\%$ . This is larger than the 10% tolerance and a discrepancy is reported.

**Figure 4-168. LVS Split Gate Ratio, Example 3**

# LVS Strict Subtypes

Specification statement

## LVS STRICT SUBTYPES {NO | YES}

### Parameters

- **NO**

Keyword that instructs LVS *not* to report discrepancies when instances with unspecified component subtypes are matched to instances with specified subtypes. This is the default behavior.

- **YES**

Keyword that instructs LVS to report discrepancies when instances with unspecified component subtypes are matched to instances with specified subtypes.

### Description

Instructs LVS applications to control the processing of component subtypes based on whether unspecified component subtypes are matched to instances with specified subtypes.

Devices are first matched without using subtypes. If this initial match is ambiguous, then subtypes are used to resolve such ambiguities.

If both LVS Strict Subtypes NO and [LVS Exact Subtypes](#) NO are specified, then a device with no subtype specified can be matched to a corresponding device of the same type and *any* subtype, empty or non-empty, without an error.

If LVS Strict Subtypes YES is specified, then the empty subtype is considered a unique subtype and the tool attempts to match it exactly. A mismatch causes a discrepancy.

LVS Exact Subtypes YES forces LVS Strict Subtypes YES, regardless of the LVS Strict Subtypes setting.

You can specify this statement once in your rule file.

### Examples

Given these two netlists:

```
Layout:      R1 1 2 100      $$ No subtype specified.
Source:      R1 1 2 100 $[RTYPE1]    $$ Subtype RTYPE1.
```

The rule file statement LVS Strict Subtypes YES reports the following discrepancy:

```
-----
21      R1   R                               R1   R(RTYPE1)
          bad component subtype
-----
```

## LVS Summary Report

Specification statement

### LVS SUMMARY REPORT *filename*

Used only in Calibre nmLVS-H.

#### Parameters

- *filename*

A required filename of an LVS Summary Report.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

#### Description

Specifies an LVS Summary Report is generated for a hierarchical LVS (not flat) run. This statement may be specified once in your rule file and is generated in addition to the [LVS Report](#). In order for the summary report to be generated, the [Mask SVDB Directory](#) QUERY keyword must also be specified.

The LVS Summary Report is generated at the end of the circuit comparison module (not the connectivity extraction module alone). The report contains information stored in the SVDB directory about both connectivity extraction and netlist comparison phases of a verification run. A complete report example is shown under “[LVS Summary Report](#)” in the *Calibre Verification User’s Manual*.

If the LVS Summary Report has been generated, this annotation appears at the end of the run transcript:

```
--- LVS SUMMARY REPORT FILE = lvs.summary
```

#### Examples

The QUERY keyword must be specified as shown in order for a summary report to be generated.

```
MASK SVDB DIRECTORY svdb QUERY
LVS SUMMARY REPORT "lvs.summary"
```

# LVS Write Injected Layout Netlist

Specification statement

## LVS WRITE INJECTED LAYOUT NETLIST “*pathname*”

Used only in Calibre nmLVS-H and PERC.

### Parameters

- “*pathname*”

A required pathname of the output SPICE netlist.

The *pathname* parameter can contain environment variables. For information regarding the use of environment variables in the *pathname* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

If the *pathname* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

### Description

Specifies to write a transformed SPICE layout netlist with injected logic, as created by the [LVS Inject Logic YES](#) statement, to the specified *pathname*. Each occurrence of an injected logic component is represented with a subcircuit call, and subcircuit definitions of injected logic circuits are written out at the end of the netlist. The netlists are written out from the LVS circuit comparison module.

This statement may appear once. It can appear together with the [LVS Write Layout Netlist](#) statement, in which case both transformed netlists, before and after logic injection, are written out. This statement is typically used only for debugging purposes. If LVS Inject Logic NO is specified, then the output of LVS Write Injected Layout Netlist is empty.

To the extent possible, the netlists preserve net, device, and instance names from the original design, except that a letter designating the SPICE element type (like M, R, C, X, and so forth) is added in front of device and instance names in the output netlist.

When the original design hierarchy is expanded, the output netlist may contain hierarchical pathnames with '/' characters serving as delimiters. The transformed netlist with injected logic preserves type, subtype, and property information, and can contain several subcircuit definitions for the same type of injected logic, but with different values of types, subtypes, or properties. A unique subcircuit name is created for each definition.

Each netlist begins with a number of header lines, as shown next.

```
** CALIBRE LAYOUT NETLIST WITH INJECTED LOGIC **
** CALIBRE VERSION: v2004.3_0.1 Wed Apr 7 18:37:17 PDT 2004
** LAYOUT NAME: lay.net (CHIP)
** NETLIST FILE: xlay.net
** GENERATED: Thu Aug 5 01:53:58 2004
```

## LVS Write Injected Layout Netlist

---

A transformed netlist with logic injection should always compare correctly with the original netlist it was generated from, provided that the original netlist was free of input errors (such as missing properties or SPICE syntax errors).

Comparing two transformed netlists should yield essentially the same results as comparing the two original netlists they were generated from, provided that the same hcells are used.

Comparing one transformed netlist and one original netlist should yield essentially the same results as comparing the two original netlists, provided that the same hcells are used. Note that in both these cases, cells representing injected logic are not used as hcells and are expanded (and then reinjected if logic injection is enabled).

However, comparing two transformed netlists with injected logic declared as hcells may yield substantially different results from the comparison of the two original netlists, for two reasons:

- Injected logic of the same topology, but with different component types, subtypes, or properties, can be matched to each other (type, subtype, or property mismatches are then reported as needed). In effect, many-to-many correspondence is possible for injected logic. However, in the transformed netlists, this logic is represented by different subcircuit definitions where many-to-many correspondence is not possible. Therefore, it may not be possible to create an hcell list that would produce the same results as the comparison of the original netlists.

For example, property errors in the comparison of the original netlists are likely to become connectivity errors in the comparison of the transformed netlists.

- Injected logic circuits can have more complex pin swappability rules than hcells. Some types of pin swappability, specifically, multiple-level swappability, cannot be propagated from the level of injected logic to that of hcells.

If this statement is used in a Calibre PERC run and [PERC Netlist](#) SOURCE is specified, then the output netlist is empty.

See also [LVS Write Injected Source Netlist](#).

## Examples

```
LVS INJECT LOGIC YES  
  
// Write the injected logic for the layout.  
LVS WRITE INJECTED LAYOUT NETLIST "injected_layout.spi"
```

# LVS Write Injected Source Netlist

Specification statement

## LVS WRITE INJECTED SOURCE NETLIST “*pathname*”

Used only in Calibre nmLVS-H and PERC.

### Parameters

- “*pathname*”

A required filename of the output SPICE netlist.

The *pathname* parameter can contain environment variables. For information regarding the use of environment variables in the *pathname* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “Key Concepts”.

If the *pathname* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

### Description

Specifies to write a transformed SPICE source netlist with injected logic, as created by the [LVS Inject Logic YES](#) statement, to the specified *pathname*. Each occurrence of an injected logic component is represented with a subcircuit call, and subcircuit definitions of injected logic circuits are written out at the end of the netlist. The netlists are written out from the LVS circuit comparison module.

This statement may appear once. It can appear together with the [LVS Write Source Netlist](#) statement, in which case both transformed netlists, before and after logic injection, are written out. This statement is typically used only for debugging purposes. If LVS Inject Logic NO is specified, then the output of LVS Write Injected Source Netlist is empty.

To the extent possible, the netlists preserve net, device, and instance names from the original design, except that a letter designating the SPICE element type (like M, R, C, X, and so forth) is added in front of device and instance names in the output netlist.

When the original design hierarchy is expanded, the output netlist may contain hierarchical pathnames with '/' characters serving as delimiters. The transformed netlist with injected logic preserves type, subtype, and property information, and can contain several subcircuit definitions for the same type of injected logic, but with different values of types, subtypes, or properties. A unique subcircuit name is created for each definition.

Each netlist begins with a number of header lines, as shown next.

```
** CALIBRE SOURCE NETLIST WITH INJECTED LOGIC **
** CALIBRE VERSION: v2004.3_0.1 Wed Apr 7 18:37:17 PDT 2004
** SOURCE NAME: src.net (CHIP)
** NETLIST FILE: xsrsrc.net
** GENERATED: Thu Aug 5 01:53:58 2004
```

A transformed netlist with logic injection should always compare correctly with the original netlist it was generated from, provided that the original netlist was free of input errors (such as missing properties or SPICE syntax errors).

Comparing two transformed netlists should yield essentially the same results as comparing the two original netlists they were generated from, provided that the same hcells are used.

Comparing one transformed netlist and one original netlist should yield essentially the same results as comparing the two original netlists, provided that the same hcells are used. Note that in both these cases, cells representing injected logic are not used as hcells and are expanded (and then reinjected if logic injection is enabled).

However, comparing two transformed netlists with injected logic declared as hcells may yield substantially different results from the comparison of the two original netlists, for two reasons:

- Injected logic of the same topology, but with different component types, subtypes, or properties, can be matched to each other (type, subtype, or property mismatches are then reported as needed). In effect, many-to-many correspondence is possible for injected logic. However, in the transformed netlists, this logic is represented by different subcircuit definitions where many-to-many correspondence is not possible. Therefore, it may not be possible to create an hcell list that would produce the same results as the comparison of the original netlists.

For example, property errors in the comparison of the original netlists are likely to become connectivity errors in the comparison of the transformed netlists.

- Injected logic circuits can have more complex pin swappability rules than hcells. Some types of pin swappability, specifically, multiple-level swappability, cannot be propagated from the level of injected logic to that of hcells.

If this statement is used in a Calibre PERC run and [PERC Netlist](#) LAYOUT is specified, then the output netlist is empty.

See also [LVS Write Injected Layout Netlist](#).

## Examples

```
LVS INJECT LOGIC YES  
  
// Write the injected logic for the source.  
LVS WRITE INJECTED SOURCE NETLIST "injected_source.spi"
```

# LVS Write Layout Netlist

Specification statement

## LVS WRITE LAYOUT NETLIST “*pathname*”

### Parameters

- “*pathname*”

A required filename of the output SPICE netlist.

The *pathname* parameter can contain environment variables. For information regarding the use of environment variables in the *pathname* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “Key Concepts”.

If the *pathname* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

### Description

Specifies to output a transformed SPICE netlist from the layout. This statement is typically used only for debugging purposes.

The netlist is written to the specified *pathname*. It represents the connectivity of the design as seen by the circuit comparison module after non-corresponding cell expansion, series and parallel device reduction, unused device filtering, and other transformations, but before logic gate recognition.

Hierarchical LVS writes a hierarchical netlist, subject to the existence of hcells. Flat LVS writes a flat netlist. To the extent possible, the netlists preserve net, device, and instance names from the original design, except that a letter designating the SPICE element type (such as M, R, C, X, and so forth) is added in front of device and instance names in the output netlist. When the original design hierarchy is expanded, then the output netlist may contain hierarchical pathnames with slash (/) characters serving as delimiters. For example:

```
MM1 1 2 3 4 p      //Original M1
MX2/X1/M1 1 2 3 4 p //Original X2/X1/M1 (down the hierarchy)
XX3 1 2 XYZ        //Original X3
XX4/X1/X1 1 2 XYZ  //Original X4/X1/X1 (down the hierarchy)
```

This statement may appear once. It operates in all flat and hierarchical operations. Netlists are written in the extended SPICE format used in Calibre nmLVS and ICtrace, with extensions coded as comments.

LVS breaks long lines of SPICE information with the continuation character (+).

Each netlist created has a header attached similar to the following example:

```
** CALIBRE LAYOUT NETLIST **
** CALIBRE VERSION: v8.7_17.1 Fri Jun 4 14:48:36 PDT 1999
** LAYOUT NAME: layout.spi(SR2F_rdc_buffer_L)
** NETLIST FILE: layout.spi.out
** GENERATED: Wed Jun 9 14:52:49 1999
```

## LVS Write Layout Netlist

---

See also [LVS Write Source Netlist](#) and [LVS Write Injected Layout Netlist](#).

### Examples

```
// Write the layout netlist after most LVS processing
// but before gate recognition.
LVS WRITE LAYOUT NETLIST "/usr/mystuff/netlists/layout1.spi"
```

# LVS Write Source Netlist

Specification statement

## LVS WRITE SOURCE NETLIST “*pathname*”

### Parameters

- “*pathname*”

A required filename of the output SPICE netlist.

The *pathname* parameter can contain environment variables. For information regarding the use of environment variables in the *pathname* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “Key Concepts”.

If the *pathname* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

### Description

Specifies to output a transformed SPICE netlist from the schematic. This statement is typically used only for debugging purposes.

The netlist is written to the specified *pathname*. It represents the connectivity of the design as seen by the circuit comparison module after non-corresponding cell expansion, series and parallel device reduction, unused device filtering, and other transformations, but before logic gate recognition.

Hierarchical LVS writes a hierarchical netlist, subject to the existence of hcells. Flat LVS writes a flat netlist. To the extent possible, the netlists preserve net, device and instance names from the original design, except that a letter designating the SPICE element type (such as M, R, C, X, and so forth) is added in front of device, and instance names in the output netlist. When the original design hierarchy is expanded then the output netlist may contain hierarchical pathnames with slash (/) characters serving as delimiters. For example:

```
MM1 1 2 3 4 p //Original M1
MX2/X1/M1 1 2 3 4 p //Original X2/X1/M1 (down the hierarchy)
XX3 1 2 XYZ //Original X3
XX4/X1/X1 1 2 XYZ //Original X4/X1/X1 (down the hierarchy)
```

This statement may appear once. It operates in all flat and hierarchical operations. Netlists are written in the extended SPICE format used in Calibre nmLVS and ICtrace, with extensions coded as comments.

LVS breaks long lines of SPICE information with the continuation character (+).

Each netlist created has a header attached similar to the following example:

```
** CALIBRE SOURCE NETLIST **
** CALIBRE VERSION: v8.7_17.1 Fri Jun 4 14:48:36 PDT 1999
** SOURCE NAME: src.spi(TOP)
** NETLIST FILE: src.spi.out
** GENERATED: Wed Jun 9 14:52:45 1999
```

## LVS Write Source Netlist

---

See also [LVS Write Layout Netlist](#) and [LVS Write Injected Source Netlist](#).

### Examples

```
// Write the source netlist after most LVS processing  
// but before gate recognition.  
  
LVS WRITE SOURCE NETLIST "/usr/mystuff/netlists/source1.spi"
```

# Magnify

Layer operation

**MAGNIFY layer BY number**

## Parameters

- **layer**  
A required original layer or a derived polygon layer.
- **BY number**  
A required keyword set, where **number** is a positive real number that specifies a magnification factor.

## Description

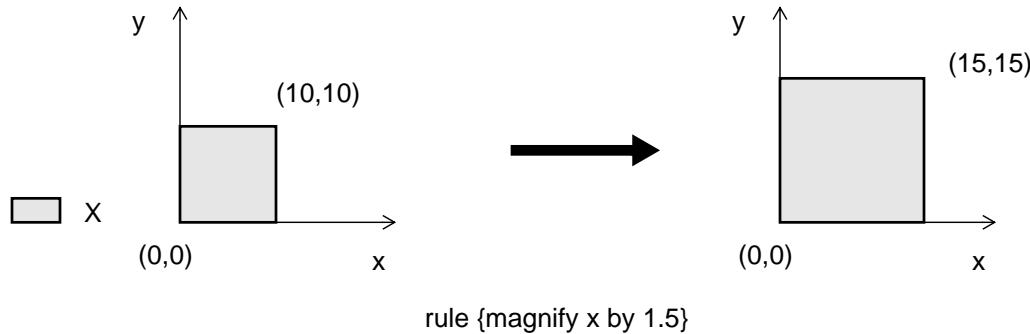
Generates a derived polygon layer by magnifying the shapes on the input **layer**. Every polygon vertex ( $x, y$ ) on **layer** is mapped to  $(x * \text{number}, y * \text{number})$ . An empty input layer produces an empty output layer.

This operation does not check coordinate space overflow. It is performed partially hierarchically in hierarchical Calibre applications.

See also [DRC Magnify Results](#) and [Layout Magnify](#).

## Examples

This example shows how magnification occurs.



# Mask Results Database

Specification statement

## MASK RESULTS DATABASE NONE

**MASK RESULTS DATABASE** *filename* [NOPROBE] [NOCONTACT] [NOLUMPED]

### Parameters

- **NONE**

Keyword that instructs the tool not to create a mask results database. This is the default behavior when you do not include this statement in the rule file.

- ***filename***

A required filename of the mask results database, if a mask results database is desired.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

- **NOPROBE**

An optional keyword that instructs the tool to exclude location probing information in the mask results database. Location probing requires extra disk space for storage and extra swap space for generation, but in most cases that extra space is relatively minor (normally not more than an additional 15%).

- **NOCONTACT**

An optional keyword that instructs the tool to exclude shapes from contact layers from the mask results database. A contact layer is any layer that appears as a contact layer in a [Connect](#) By operation. This option saves disk space and execution time. The penalty is contact layers are not shown by Pyxis Layout **show** commands and you cannot show nets by location by specifying a contact layer.

- **NOLUMPED**

An optional keyword that instructs the tool to exclude lumped parasitics from the mask results database.

### Description

Specifies the ICtrace mask results database filename and the type of information to exclude from the mask results. You can specify this statement once.

The file produced by this statement is used in ICtrace for LVS discrepancy debugging; the file is not used by Calibre applications. In Pyxis Layout, this statement sets the value of the ICtrace application variable `mask_default_database_name`.

Although Calibre nmLVS can produce a Mask Results Database, this database cannot be loaded into Calibre RVE. For results that are usable in RVE, see [Mask SVDB Directory](#).

## **Examples**

```
// ICtrace Mask results database  
MASK RESULTS DATABASE "maskdb"
```

## Mask SVDB Directory

Specification statement

### MASK SVDB DIRECTORY *filename*

[QUERY] [XRC | XACT] [CCI] [SI] [IXF] [NXF] [PHDB] [PINLOC | NOPINLOC]  
[GDSII] [XDB] [DV] [SLPH] [NETLIST] [ANNOTATE DEVICES] [NOFLAT]  
[BY GATE] [OMN]

Used only in Calibre. Required for Calibre xACT, Calibre xRC, and Calibre RVE.

### Summary

Specifies the types of files generated in the standard verification database (SVDB) by nmLVS, PEX, and PERC applications.

### Parameters

- *filename*

A required absolute or relative filename of a directory.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

- QUERY

An optional keyword that creates all information necessary for full execution of Calibre nmLVS RVE, Calibre PERC RVE, and the Query Server (calibre -query), with the exception of the Calibre Connectivity Interface (CCI) subset of Query Server commands.

The QUERY option creates information described in the PHDB, XDB, and DV options. Pin location information is not generated by the QUERY keyword. To enable Connectivity Interface commands, specify the CCI option. See the [Calibre Query Server Manual](#) for more information about these topics.

- XRC

Creates all information necessary for the Calibre xRC flow. This option creates information described in the PHDB and XDB options. In addition, this option ensures that all layers participating in the Calibre xRC flow receive valid node numbers (when required) and that these layers are stored in the PHDB database. Specifically, this option ensures valid node numbers and storage for relevant layers from PEX specification statements. If [LVS Inject Logic YES](#) is not explicitly specified, XRC sets it to NO. This option has no effect in flat Calibre nmLVS.

SPICE netlists extracted with calibre -spice when the XRC option is specified may differ somewhat from SPICE netlists extracted without that option. The netlists are equivalent but node numbers may differ. The difference is that, in some cases, more layers may participate in connectivity extraction when the XRC option is specified.

The XRC option in itself does not create all information necessary for full execution of nmLVS RVE or the Query Server (calibre -query). If you intend to use nmLVS RVE or the

non-CCI subset of Query Server commands, then you should specify the QUERY option in addition to the XRC option. If you intend to use the CCI subset of Query Server commands, then you should specify the CCI option in addition to the XRC option.

This option disables the [LVS Push Devices](#) LDE option. XRC may not be specified with XACT.

- **XACT**

Creates all information necessary for the Calibre xACT flow. This option creates information described in the PHDB and XDB options. In addition, this option ensures that all layers participating in the Calibre xACT flow receive valid node numbers (when required) and that these layers are stored in the PHDB database. Specifically, this option ensures valid node numbers and storage for relevant layers from PEX specification statements. If [LVS Inject Logic YES](#) is not explicitly specified, XACT sets it to NO. This option has no effect in flat Calibre nmLVS.

SPICE netlists extracted with calibre -spice when the XACT option is specified may differ somewhat from SPICE netlists extracted without that option. The netlists are equivalent but node numbers may differ. The difference is that, in some cases, more layers may participate in connectivity extraction when the XACT option is specified.

The XACT option in itself does not create all information necessary for full execution of nmLVS RVE or the Query Server (calibre -query). If you intend to use nmLVS RVE or the non-CCI subset of Query Server commands, then you should specify the QUERY option in addition to the XACT option. If you intend to use the CCI subset of Query Server commands, then you should specify the CCI option in addition to the XACT option.

This option disables the [LVS Push Devices](#) LDE option. XACT may not be specified with XRC.

- **CCI**

An optional keyword that creates all information necessary for executing the full Calibre Connectivity Interface subset of Query Server commands. This option creates information described in the PHDB, GDSII, XDB, NETLIST, and ANNOTATE DEVICES options. It triggers the PINLOC option, unless you specify NOPINLOC. In the default mode, the CCI mode prevents device pushdown. See “[Situations Preventing Pushdown](#)” on page 974.

The CCI option reserves a Calibre Connectivity Interface (CI) product license during execution of circuit extraction in Calibre nmLVS (flat and hierarchical). If LVS Inject Logic is unspecified, then the CCI option causes the logic injection to be disabled.

This option is not designed to provide access to Calibre RVE for LVS functionality or to non-CCI Query Server commands. To gain full access to Calibre RVE for LVS and to non-CCI Query Server commands, specify the QUERY option also.

This option disables the [LVS Push Devices](#) LDE option.

- **SI**

An optional keyword that creates all the information necessary for running interactive short isolation. This option provides the necessary files regardless of the [LVS Isolate Shorts](#)

setting in the rule file. It is useful only in a hierarchical connectivity extraction (calibre -spice) run.

This option produces a READ/WRITE persistent hierarchical database (PHDB) containing geometric data and a *layout\_primary.extf* file. Additional files are written to the SVDB directory as you run short isolation in the interactive mode. For additional information see the section called “[Using Short Database Commands](#)” in the *Calibre Query Server* manual.

This option also enables interactive short verification in Calibre RVE for LVS; see “[Verifying Short Repairs](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

The PHDB directory created with this option is in a read-write format that must not be altered. If you archive the directory to a read-only environment, it will not open. Therefore, you should not use this option when you intend to create a read-only archive of the SVDB. The directory can only be copied successfully using the qs::copy\_svdb command in the Query Server Tcl shell. Operating system commands such as cp and tar should not be used for copying an SVDB created with the SI option.

- IXF  
An optional keyword that specifies to create an ASCII instance cross-reference file, *layout\_primary.ixf*. For an example of the output, refer to the “[SVDB Cross-Reference Files](#)” in the *Calibre Verification User’s Manual*.
- NXF  
An optional keyword that creates an ASCII net cross-reference file, *layout\_primary.nxf*. For an example of the output, refer to the “[SVDB Cross-Reference Files](#)” in the *Calibre Verification User’s Manual*.
- PHDB  
An optional keyword that creates a persistent hierarchical database (PHDB) with layout shapes and layout connectivity and device information called *layout\_primary.phdb*.  
This PHDB database contains sufficient information for probing the layout with RVE or the Query Server (calibre -query). It contains no source information and is not sufficient for cross-probing between layout and source. Such a PHDB database is created by the QUERY option. You may use the PHDB option explicitly instead of QUERY to avoid creating the XDB cross-reference database, which is sometimes time-consuming to generate, particularly when you execute circuit comparison with no corresponding cells (hcells).  
The PHDB option by itself does not allow you to execute the Calibre Connectivity Interface (CCI) subset of Query Server commands; to do that, specify the CCI option or relevant specialized options. Note that pin location information is not generated unless the PINLOC option is specified or triggered.
- PINLOC | NOPINLOC  
Optional keywords that control generation of pin location information. You may not specify both simultaneously. Pin location information is required by some third-party applications. It is not used in typical LVS runs.

**PINLOC** — Triggers generation of pin location information in the PHDB. This behavior is enabled by default when you specify the CCI option. You would use PINLOC most commonly with the QUERY or PHDB options if you need pin locations in addition to the default data that these latter options provide. PINLOC is ignored in LVS. You may not specify this keyword with NOPINLOC.

**NOPINLOC** — Suppresses generation of pin location information in the PHDB if the CCI option is also used.

Requesting pin location information affects device pushdown. See “[Situations Preventing Pushdown](#)” on page 974.

- **GDSII**

An optional keyword that creates information sufficient for generating Annotated GDSII Files (AGF) from the SVDB database. This is done in the Query Server with the GDS WRITE command, which is part of the Calibre Connectivity Interface set of commands. The GDSII option reserves a Calibre Connectivity Interface (CI) product license during execution of circuit extraction in Calibre nmLVS (flat and hierarchical).

The GDSII option creates a PHDB database with layout shapes, layout connectivity, and device information (as in the PHDB option), plus fully-merged device seed shapes annotated with device numbers (as in the ANNOTATE DEVICES option). It does not create a cross-reference database (as in the CCI or XDB options); thus, the resulting SVDB database is smaller than what would be created with the CCI option.

- **XDB**

An optional keyword that creates a binary cross-reference database, *layout\_primary.xdb*. The XDB contains cross-reference information between layout and source elements as established in the LVS circuit comparison step. The QUERY, CCI, XACT, and XRC keywords also trigger the XDB keyword by default.

- **DV**

An optional keyword that creates a Discrepancy Viewer database for the nmLVS RVE discrepancy viewer. This is a text file called *layout\_primary.dv*. It contains formatted information corresponding to that in the LVS report.

- **SLPH**

An optional keyword that creates two ASCII files: a layout placement hierarchy file *layout\_primary.lph* and a source placement hierarchy file *layout\_primary.sph*. This option is ignored in flat LVS.

- **NETLIST**

An optional keyword that creates information sufficient for generating layout netlists from the SVDB database. This is done in the Query Server with the LAYOUT NETLIST WRITE command, which is part of the Calibre Connectivity Interface set of commands. The generated netlists may contain vertex locations for devices. To allow center locations, specify the ANNOTATE DEVICES option in addition to the NETLIST option. The generated netlists may contain layout names or layout IDs for nets and instances. To allow

source names, specify the XDB option in addition to the NETLIST option. The NETLIST option reserves a Calibre Connectivity Interface (CI) product license during execution of circuit extraction in Calibre nmLVS (flat and hierarchical).

- **ANNOTATE DEVICES**

An optional keyword that adds fully-merged device seed shapes annotated with device numbers to the PHDB database, if one is created. This information is required by the Query Server LAYOUT NETLIST WRITE command when center locations of devices are requested (Query Server command LAYOUT NETLIST DEVICE LOCATION CENTER). The ANNOTATE DEVICES option is intended for use in conjunction with the NETLIST option. By itself, ANNOTATE DEVICES does not create a PHDB database and it has no effect unless a PHDB database is created.

- **NOFLAT**

An optional keyword that instructs flat nmLVS *not* to create the SVDB directory or any associated files. This option can be useful if you intend to run a large design through flat LVS. Creating an SVDB for such a run can result in huge files and long runtimes. It may be more beneficial to get the flat LVS results without the overhead of creating an SVDB in these cases.

- **BY GATE**

An optional keyword that instructs Calibre nmLVS applications to write information about logic gates into the instance cross reference file (.ixf), if one is created. For an example of the output, refer to the “[SVDB Cross-Reference Files](#)” in the *Calibre Verification User’s Manual*.

- **OMN**

An optional keyword that adds original model names to the XDB database, if one is created. Original model names are names of devices as they appear in an input SPICE netlist before any processing of \*.EQUIV statement substitutions. This option is only used by hierarchical LVS; flat LVS ignores it.

### Description

Specifies a Standard Verification Database (SVDB) directory that stores data files and directories created by Calibre nmLVS/nmLVS-H for use by Calibre nmLVS RVE and the Query Server. It also stores files and directories created by the Calibre xRC flow and Calibre xACT flow.



#### Note

Flat Calibre nmLVS generates a SVDB database only if the Mask SVDB Directory statement contains the QUERY, PHDB, or XDB optional keywords.

---

Zero or more options may be specified. Options may be combined and may be redundant in terms of the outputs produced. For example, the following statement is required when you use the Query Server in a Calibre xRC flow:

```
MASK SVDB DIRECTORY svdb QUERY XRC
```

The QUERY and XRC options produce similarly-named output files.

All files are created in the specified SVDB directory. The SVDB directory is created if it does not exist. Files are overwritten if they already exist. The *layout\_primary* component in file names is the top-level cell name specified with the [Layout Primary](#) specification statement. If no Layout Primary is specified, then the name ICV\_UNNAMED\_TOP is used.

In Calibre nmLVS-H, you can execute circuit extraction and circuit comparison together or separately; SVDB output is valid in both cases, but extraction and comparison do not produce the same outputs. The PHDB database is created during LVS circuit extraction (calibre -spice, when explicitly specified in nmLVS-H). The .xdb, .ixf, .nxn, .lph and .sph files are created during LVS circuit comparison when the appropriate keywords are specified.

Names of files generated during circuit extraction and circuit comparison are stored in two files in the SVDB directory. Names of files generated during circuit extraction are written to the *topcell.extf* file, and files generated during circuit comparison are written to the *topcell.lvsf* file, where *topcell* is the name of the top-level cell specified with Layout Primary.

Calibre nmLVS RVE uses information in these files to populate the file names in the files pane.

Note that some Query Server commands will not operate in SVDB databases created by flat LVS. For example, commands that involve the concepts of cells and cell placements produce error messages.

The permission structure of the PHDB directory must not be altered. Specifically, locked files are created inside the directory structure. If the locked files cannot be created, the PHDB cannot be opened.

To copy an SVDB directory, use the [qs::copy\\_svdb](#) command from the Query Server Tcl Shell. The “cp -r” and “mv” operating system commands can corrupt the SVDB, but the “tar” command may be used so long as the archive is created when the database is not being altered. To remove an SVDB directory without a prompt, use “rm -rf”. An SVDB created using the SI option should not be placed in a read-only environment because the database will not open.

When a Mask SVDB Directory statement is present in the rule file, the Calibre command line options -ixf and -nxn write their output to the indicated SVDB directory and use SVDB-style file names indicated above instead of using the [LVS Report](#) file name as a prefix.

In addition to a SVDB database, flat Calibre nmLVS may also generate a maskdb database if one is specified with the [Mask Results Database](#) specification statement. These databases are not related.

## Summary of Database Files

The following files and directories are created and stored when the QUERY option is specified. They allow you to use the Calibre Results Viewing Environment to query results.

persistent hierarchical database (PHDB)	<i>layout_primary.phdb</i>
cross-reference information file (XDB)	<i>layout_primary.xdb</i>
discrepancy viewer file (for RVE)	<i>layout_primary.dv</i>

The CCI, QUERY, XRC, and XACT keywords cause all three of these files to be generated.

Table 4-29 shows the files created by each switch for the Mask SVDB Directory specification statement:

**Table 4-29. Files Generated by Mask SVDB Directory Options**

Option	.dv	.xdb	.phdb	.ixf	.nxf	.sph	.lph
QUERY	Y	Y	Y				
XRC	Y	Y	Y				
XACT	Y	Y	Y				
CCI	Y	Y	Y				
PHDB	Y		Y				
XDB	Y	Y					
DV	Y						
SI			Y				
GDSII			Y				
IXF				Y			
NXF					Y		
SLPH						Y	Y

The contents of the files shown in this table are not necessarily identical. See the option descriptions for details about usage.

The cross-reference database (XDB) file is not written if [LVS Report Option XR](#) is specified and the result is CORRECT.

See “[SVDB Cross-Reference Files](#)” in the *Calibre Verification User’s Manual* for a discussion of cross-reference files.

## QUERY Details

You must specify QUERY if you plan to use all aspects of the Calibre Results Viewing Environment (RVE). When QUERY is set, Calibre nmLVS/nmLVS-H generates the file *layout\_primary.dv*. This file is required for RVE. You can run RVE in a limited form by specifying the PHDB or XDB options independently.

See the [Calibre Query Server Manual](#) for more information about the Query Server and the Calibre Connectivity Interface.

## XRC Details

The XRC option must be used when you intend to run a Calibre xRC flow. The XRC keyword causes Calibre nmLVS-H to produce a Calibre xRC compatible layout netlist and a PHDB that can be used with the Query Server for Calibre xRC debugging queries. The XRC option also

causes Calibre nmLVS-H to produce an XDB optimized for Calibre xRC use. Other files and directories created as a product of the Calibre xRC flow are not affected, although they are also stored in the SVDB. This includes the parasitic database (PDB). Calibre nmLVS RVE can read the PDB and display extracted parasitic data.

## XACT Details

The XACT option must be used when you intend to run a Calibre xACT flow. The XACT keyword causes Calibre nmLVS-H to produce a Calibre xACT compatible layout netlist and a PHDB that can be used with the Query Server for Calibre xACT debugging queries. The XACT option also causes Calibre nmLVS-H to produce an XDB optimized for Calibre xACT use. Other files and directories created as a product of the Calibre xACT flow are not affected, although they are also stored in the SVDB. This includes the parasitic database (PDB). Calibre nmLVS RVE can read the PDB and display extracted parasitic data.

## PHDB Details

The acronym PHDB stands for Persistent Hierarchical DataBase. This is a directory called *layout\_primary.phdb*. The PHDB contains layout connectivity and device information, both from the LVS circuit extraction step. Layout connectivity and device information is generally sufficient for obtaining lists of layout nets and devices and for similar queries. Source information is stored separately and is not present in a PHDB. Thus, a PHDB is never sufficient for cross-probing between layout and source. The PHDB database is in binary form. Client access is provided through the Query Server (calibre -query or -qs) or LVS-RVE.

In Calibre nmLVS-H, this information is created during hierarchical circuit extraction. In flat Calibre nmLVS, this information is created during the flat connectivity extraction and device recognition steps. Layout shapes are generally sufficient for highlighting layout nets and devices.

Subject to options specified, a PHDB database may contain the following information:

- Cell and cell placement information (hierarchical mode only).
- Geometry information for layers that appear in [Connect](#) or [Sconnect](#) operations; serve as [Device](#) seed, pin, or auxiliary layers; or are target layers of [Stamp](#) operations (second argument of Stamp).
- The complete text of the rule file(s) used for circuit extraction.
- Summary extracted device information including device coordinates, types, calculated properties, pin and pin location information.
- Net connectivity information through the hierarchy of cell placements.
- Optionally, device seed shape information can be saved for access by the Query Server's Calibre Connectivity Interface (CCI) commands by using the CCI or ANNOTATE DEVICES options.

## XDB Details

The XDB option writes a cross-reference database (XDB). The XDB contains sufficient information for performing cross-reference related functions within RVE or the Query Server, for example retrieving the source name of a layout element or vice-versa. It contains no geometrical information. The XDB database is in binary form; the size is limited by the resources of the host platform. Client access is provided through the Query Server or RVE.

The XDB database contents are these:

- Net cross reference for nets matched during circuit comparison, including nets matched to multiple corresponding nets.
- Incorrect nets and unmatched nets.
- Instance cross reference for instances matched during circuit comparison, including reduced instances and logic gate membership information.
- Unmatched instances (except filtered instances).
- Original netlist placement hierarchy information (for hierarchical mode only).

Note that creation of a XDB database in an SVDB directory causes the removal of any existing ASCII cross reference files (.ixf, .nxr, .lph, and .sph) with the same *layout\_primary* name that may be present in that SVDB directory.

## NETLIST Details

The NETLIST option creates a PHDB database with connectivity information but little or no geometry information. Such PHDB database is often significantly smaller than one created with the PHDB or QUERY options, but it is only sufficient for executing a limited subset of Query Server commands (LAYOUT NETLIST WRITE and various configuration commands).

Note that in order to generate layout netlists from the SVDB database, you must specify the NETLIST option (or the CCI option, which includes it) even if you specify the PHDB or QUERY options.

You can specify the Mask SVDB Directory in Calibre Interactive—nmLVS. See “[Specifying Outputs for an nmLVS Run](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

## SVDB Compatibility Across Calibre Versions

An SVDB generated with an earlier version of Calibre nmLVS may generally be used in a later version of Calibre. The only exceptions are Calibre nmLVS versions 2004.3 and 2004.4, which did not use earlier SVDB versions. Note that SVDBs from earlier Calibre nmLVS versions may not take advantage of the complete set of functionality available in later versions, but the functionality from an earlier version will still apply in a later version.

The PDB file generated by Calibre xRC or Calibre xACT is only compatible with the software version that generated it. The PHDB file generated by Calibre xRC or Calibre xACT may generally be used with a later version of Calibre.

An SVDB created with a later version of Calibre tools may not generally be used in an earlier version.

## Examples

### Example 1

```
// generates the query database for RVE and the Query Server  
MASK SVDB DIRECTORY svdb QUERY
```

### Example 2

```
// generate the databases for the connectivity interface, RVE,  
// and Query Server  
MASK SVDB DIRECTORY svdb QUERY CCI
```

### Example 3

```
// generate only cross-reference and placement hierarchy files  
MASK SVDB DIRECTORY "../adder_svdb" IXF NXF SLPH
```

### Example 4

```
// generate the databases for the connectivity interface, RVE,  
// Query Server, and xRC  
MASK SVDB DIRECTORY svdb XRC QUERY CCI
```

### Example 5

```
// generate database for interactive short isolation  
MASK SVDB DIRECTORY svdb SI
```

## MDP Checkmap

Mask data prep operation

```
MDP CHECKMAP input_layer
[FILENAME name_of_file] FILE { parameter_block | '['
    [section_size sec_value]
    [output_layer [layer | layer datatype]]
    [cblock]
    [prefix string] [append string]
    [strict {0 | 1}]
    ']'
}
```

### Summary

The command outputs single or multiple layers and datatypes with bin injection for large cells.

### Parameters

- ***input\_layer***

The name of an original or derived polygon layer. This is the layer that contains the data to which Calibre applies the map-sizing functionality.

- **FILENAME *name\_of\_file***

A keyword specifying the name of the input file.

- **FILE *parameter\_block***

A keyword/argument pair specifying that the parameters are contained in a reusable parameter block defined using the [Litho File](#) SVRF statement.

- **FILE '[' ... ']**

A keyword/argument set specifying that the map size parameters are contained inline within the square brackets ( [] ). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines STRICTLY BETWEEN the left and right brackets (that is, cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **section\_size *sec\_value***

An optional keyword that specifies the size of the square section in microns for large cells. The default value is 1000 um.

- **output\_layer [layer | layer datatype]**  
Optional output *layer* or *layer* and *datatype* pair mapping. The number of the *layer datatype* pairs must match the number of input layers.
- **cblock**  
Optional keyword that specifies that the output be compressed using the CBLOCK record as defined by the OASIS format. Compression is done only when it gives an advantage in terms of amount of data.
- **prefix string**  
Optional keyword and parameter that instructs Calibre to prefix all cell names with string. This keyword is only valid for GDSII or OASIS format databases. This behavior applies only to results from Calibre nmDRC-H.
- **append string**  
Optional keyword and parameter that instructs Calibre to append all cell names with string. This keyword is only valid for GDSII or OASIS format databases. This behavior applies only to results from Calibre nmDRC-H.
- **strict {0 | 1}**  
The OASIS format has a strict mode that can be read more efficiently. This optional parameter controls whether or not the OASIS output layout file will be strict mode or not. If 1 is specified, the output will be strict mode. If 0 is specified, the output is not in strict mode. The default value is 1.

## Description

The MDP Checkmap statement is used primarily for OASIS bin injection. The command outputs single or multiple layers and datatypes with bin injection for large cells. The criteria for bin injection are based on the physical extent and geometry content of a cell.

## Examples

```
mapped = MDP CHECKMAP nw pa p1 pp na pd m2 cc m1 v1 p1_opc
        FILENAME "./flat_mapped_datatypes.oas" FILE [ \
        section_size 1000
        output_layer 1 2 6 7 8 ( 11 3 ) ( 12 4 ) ( 13 5 ) ( 14 6 ) \
        ( 15 7 ) ( 28 8 )
    ]
mapped { COPY mapped } DRC CHECK MAP mapped 99
```

## MDP Embed

Mask data prep operation

```
MDP EMBED input_layer [input_layer2...] [MAP output_layer]
[FILENAME output_file_name]
{FILE litho_file_block | FILE '[
    section_size sec_value
    maximum_output_count count
    [vboasis_path filepath [-noCheck]]
    [svrf_layer_name {layer_name | {layer_name oasis_layer_number
        [oasis_layer_datatype]...}}]
    [vboasis_precision_multiplier {n [/d] | AUTO}]
    [vboasis_layout_magnify n [d]]
    [vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
    [jobdeck path [-ejobdeck_clipping_severity {1 | 2}] [-ejobdeck_pattern_suffix suffix
        [-data_search_path path]]
    [svrf_layer_name {name | {name layer_number [datatype_number]}}...]
    [data_extent {HDB | FILE | DIRECTIN}]
    [oasis_output_path [-perSection] path]
    [oasis_output_layers {layer [datatype] | {{map_name layer [datatype]}}...}}]
    [oasis_output_primary [HDB | DIRECTIN | FILE | -explicit topcell_name]
    [oasis_output_options [-cblock] [-maximum_vertex {mv | ALL}]]]
    [direct_FS_access [INPUT | INTERMEDIATE]...]
    [<SVRFSTART>
        svrf_statements
    <SVRFEND>]
    ']
```

### Summary

For detailed information about the parameters, refer to “[MDP Embed](#)” in the *Calibre Mask Data Preparation User’s Manual*.

### Description

The MDP Embed command provides section-based processing for some SVRF commands without requiring a FRACTURE operation.

Section-based processing is a method for traversing large designs which is specifically targeted for flat hierarchy. Because of this, section-based processing is typically used for FRACTURE operations.

Hierarchical processes traverse a layout visiting by each cell, then moving on to the next cell. Hierarchical processing breaks the job into tasks for hierarchical cells. However, these cells can vary in size, and because they are non-uniform in size and have a limited number, hierarchical processing may not be able to use additional processors beyond a certain point.

In section-based processing, the chip layout is divided into a grid of blocks called “sections.” The results are computed per section, rather than per cell, improving scalability and, in some cases, improving performance (particularly for flat designs). For each section:

1. Data in that section is requested from the hierarchical database (HDB).
2. The results are computed.
3. The output is reassembled. Note that the outputs are flat.

Processing in each section can be done independently on a separate CPU. Section processing attempts to optimize for turn-around time (or elapsed time) rather than efficiency, ignoring potential redundancies when traversing cells (it flattens the hierarchy).

# MDP Mapsize

Mask data prep operation

```
MDP MAPSIZE input_layer map_layer_1 [map_layer_n ...]
{FILE parameter_block | FILE '['
  soft_boundary { 0 | sb_value }
  map_size { size_value_1 [size_value_n...]}}
  ']'
}
```

## Summary

This statement allows you to variably resize elements of your layout based on their geographical locations, which are determined by *map-size regions*. The sizing values are selected as a response to the strength of a physical process effect that requires a data-based correction. Examples are density-based process-loading effects and variation of processing by radial location within a particular machine.

## Parameters

- ***input\_layer***

The name of an original or derived polygon layer. This is the layer that contains the data to which Calibre applies the map-sizing functionality.

- ***map\_layer\_1* [*map\_layer\_n* ...]**

The name(s) of an original or derived polygon layer. You can specify this argument any number of times. These layers specify your map-size regions. The number of *map\_layer* arguments must equal the number of *size\_value* arguments to the **map\_size** keyword.

- ***FILE parameter\_block***

A keyword/argument pair specifying that the map size parameters are contained in a reusable parameter block defined using the [Litho File](#) statement.

- ***FILE* '[' ... ']**

A keyword/argument set specifying that the map size parameters are contained inline within the square brackets ( [ ] ). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines strictly between the left and right brackets (that is, cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **soft\_boundary { 0 | sb\_value }**

A keyword/argument pair that specifies the minimum size, in user units, for which Calibre creates an edge segment for edges that cross region boundaries. The default value is 0.

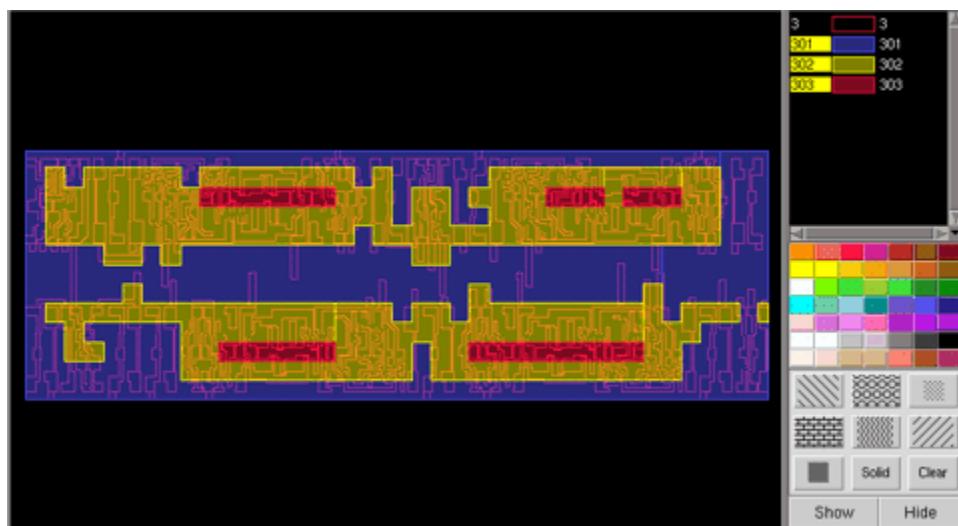
- **map\_size {size\_value<sub>1</sub> [size\_value<sub>N</sub> ...]}**

A keyword/argument set that instructs Calibre to perform a map-sizing operation on the data from *input\_layer*, where *size\_value* is the map-size value, in user units, for the respective *map\_layer*. The number of *size\_value* arguments must match the number of *map\_layer* arguments.

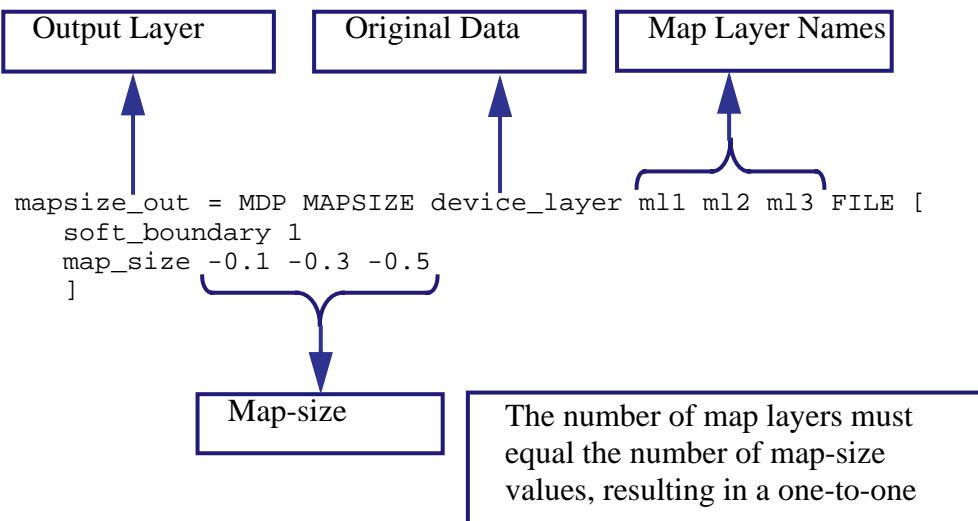
## Description

To prepare your data for map-sizing, you must partition your data into non-overlapping map-size regions by creating additional layers in your design data. These new layers are referred to as map layers. You can create these layers either manually, by drawing marker shapes in a layout editor like Calibre MDPview, or derive them inside Calibre using Density, Density Convolve, or other SVRF operations. [Figure 4-169](#) shows an example of what a design could look like with map-size regions. The graphics and examples for this section are based on the output of the Density Convolve examples.

**Figure 4-169. Data With Map Layers**



You can resize elements within each map-size region by a different value. [Figure 4-170](#) shows an example of a MDP Mapsize statement.

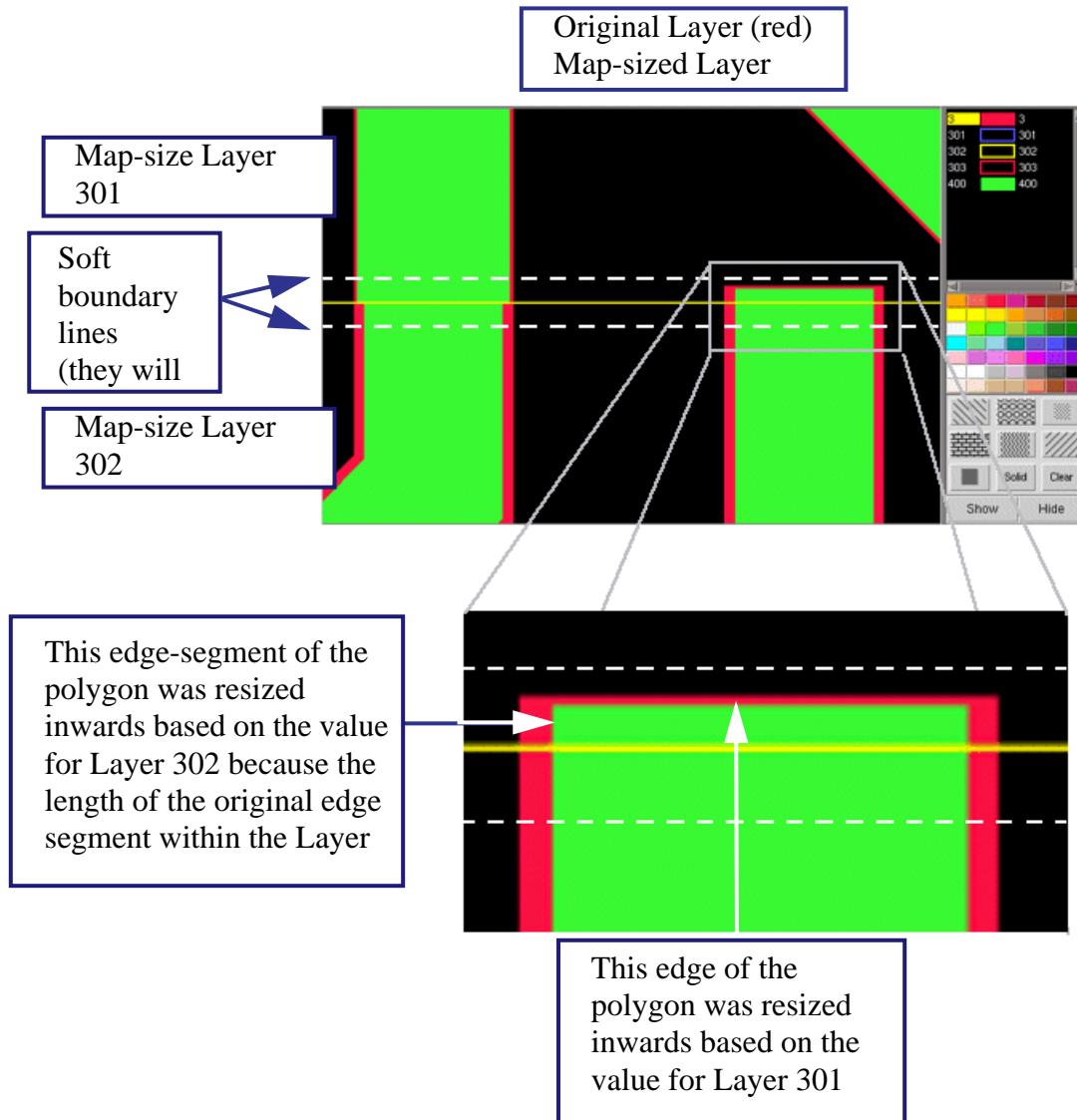
**Figure 4-170. MDP Mapsize SVRF Example**

The output layer (mapsize\_out) contains the resized data, with the map-size values applied to the edges of the elements of the original layer in accordance to their location relative to the marker layers (map-size regions).

When Calibre resizes the polygons on the original data layer (*input\_layer*) it actually resizes the individual edges of the polygons, which means that any polygon that crosses a map-size region could have edges sized by different values and in different directions, depending on whether the map-size value is positive or negative.

In order to avoid unnecessary jogs in polygons on the output layer, which allows for better lithographic performance and maintains the best possible data quality on the output of the subsequent fracture step (shot count reduction), Calibre sets the exact cutline between adjacent map-size regions within a margin around the region border. You specify this margin size with the **soft\_boundary** keyword. Any edge that is within the soft boundary is resized by the value of the region containing the majority of the edge. If a shape crosses the soft boundary on either side of the region border, the transition between the variable sizing values for each region is aligned with an existing jog in the pattern. Only when there are no existing vertices in the entire soft\_boundary area will the polygon be cut along the border. [Figure 4-171](#) shows an example of how Calibre resizes geometries that cross region boundaries and soft boundaries.

Operations such as map-size can severely degrade the hierarchy of the input layer. For this reason, map-sizing is frequently used as an embedded SVRF command in section-processing mode.

**Figure 4-171. Soft Boundary Behavior Example**

## Examples

The following is a complete rule file used for resizing the original layer (layer 3).

```
LAYOUT SYSTEM OASIS
LAYOUT PATH "convolve_ring_output.oas"
LAYOUT PRIMARY "ring"

PRECISION 1000
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "ring_mapsize_output.oas" OASIS PSEUDO

LAYER orig3 3
```

## MDP Mapsize

---

```
LAYER orig301 301
LAYER orig302 302
LAYER orig303 303

mapsize_out = MDP MAPSIZE 3 301 302 303 FILE [
soft_boundary 1
map_size -0.1 -0.3 -0.5
]

orig3 {COPY orig3 } DRC CHECK MAP orig3 3
orig301 {COPY orig301 } DRC CHECK MAP orig301 301
orig302 {COPY orig302 } DRC CHECK MAP orig302 302
orig303 {COPY orig303 } DRC CHECK MAP orig303 303
mapsize_out {COPY mapsize_out} DRC CHECK MAP mapsize_out 400
```

# MDP Maskopt

Mask data prep operation

**MDP MASKOPT** *input\_layer* [*exclude\_layer*]

```
FILE { parameter_block | '['
      [max_jog_width value]
      [min_edge_separation value]
      [max_jog_alignment value]
      [max_jog_area value]
      ']'
    }
```

## Summary

The MDP MASKOPT SVRF operation modifies the input geometry to improve fracture quality.

## Arguments

- ***input\_layer***

The name of an original or derived polygon layer.

- ***exclude\_layer***

Specifies a layer where the geometry of the ***input\_layer*** that lies partially or fully within the region defined by the ***exclude\_layer*** polygon(s) will not be modified by MDP MASKOPT.

- **FILE *parameter\_block***

A keyword/argument pair specifying that the parameters are contained in a reusable parameter block defined using the [Litho File](#) SVRF statement.

- **FILE '[' ... ']**

A keyword/argument set specifying that the map size parameters are contained inline within the square brackets ( [] ). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines STRICTLY BETWEEN the left and right brackets (that is, cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- **max\_jog\_width *value***

Specifies the maximum allowable length for an edge of a polygon to be qualified as a jog.

- **min\_edge\_separation *value***

Specifies the minimum required distance between a jog and the next nearby polygonal edge of the same orientation (if the jog is vertical, then it looks for the next vertical edge; if the

jog is horizontal, it will look for the next horizontal edge, and so on). If, by moving the jog, that distance would be violated, then the jog is not moved.

- *max\_jog\_alignment value*

Specifies the maximum allowable jog movements during alignment optimization.

- *max\_jog\_area value*

Specifies the maximum allowable area for a jog movement during alignment optimization. This is an optional parameter which may be specified in addition to the values for *max\_jog\_width* and *max\_jog\_alignment*. Its value is specified in square microns. The default value is *off*, which means that *max\_jog\_area* is equal to (*max\_jog\_width* \* *max\_jog\_alignment*). In use, this value will be set to something smaller than (*max\_jog\_width* \* *max\_jog\_alignment*). If the result of moving a jog is larger than this area, the jog will not be moved. This permits more jogs to be moved (a narrow jog with a long move for alignment can still be moved without harming image quality).

---

### Note



In general, *max\_jog\_area* should not be a great deal smaller than (*max\_jog\_width* \* *max\_jog\_alignment*). Values like 1/4 to 1/10 of (*max\_jog\_width* \* *max\_jog\_alignment*) are reasonable. Smaller values tend to use a great deal of CPU time without improving results significantly.

---

## Description

The MDP MASKOPT SVRF operation modifies the input geometry to improve fracture quality. Constrained jog movements are applied to form polygons which, when fractured, will have fewer shot counts and fewer small figures than the initial fractured polygons.

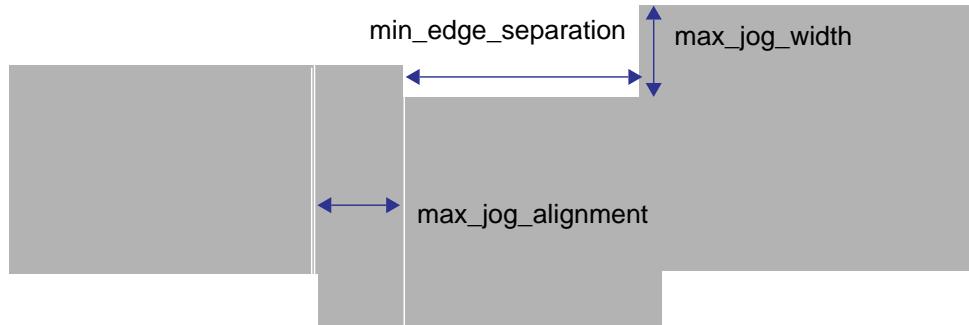
[Figure 4-172](#) illustrates where the values for *max\_jog\_width*, *max\_jog\_alignment*, and *min\_edge\_separation* are applied.

### Figure 4-172. Illustration of MDP MASKOPT Arguments

**max\_jog\_width:** The maximum allowable length for an edge of a polygon to be qualified as a jog.

**max\_jog\_alignment:** The maximum allowable jog movements during alignment optimization.

**min\_edge\_separation:** The minimum required mask geometric constraints between jogs and the nearby polygonal edges of the same orientation.



### Example

```
aligned = MDP MASKOPT poly FILE [
    max_jog_width 0.01
    min_edge_separation 0.055
    max_jog_alignment 0.055
]
aligned { COPY aligned } DRC CHECK MAP aligned 5
```

## MDP Oasis\_Extent

Mask data prep operation

### MDP OASIS\_EXTENT [MAP ID]

```
FILE { parameter_block | '['
    [layout_subset name {layer_range datatype_range} ...]
    [vboasis_path file_path [-noCheck]]
    [vboasis_precision_multiplier {n[d] | AUTO}]
    ']'
}
```

### Summary

The MDP OASIS\_EXTENT SVRF operation computes the extent of an OASIS layout.

### Arguments

- **MAP ID**

Multiple extent outputs are assigned to a particular layout\_subset using the MAP identifier. The extent of the specified layout\_subset is the output for that layout\_subset. Map identifiers must have a one-to-one, complete mapping to the specified extent scope. Only polygon-directed rule output layers are allowed.

- **FILE parameter\_block**

A keyword/argument pair specifying that the parameters are contained in a reusable parameter block defined using the [Litho File](#) SVRF statement.

- **FILE '[' ... ']'**

A keyword/argument set specifying that the map size parameters are contained inline within the square brackets ( [] ). Square brackets must surround all parameters appearing after the keyword FILE. In addition, everything within the square brackets must comply with the following requirements:

- Keywords and parameters must be on lines STRICTLY BETWEEN the left and right brackets (that is, cannot be on the same line).
- You can include comments within the section of the operation set off by square brackets, if you begin the comments with a double forward slash (//). This comment character indicates that all text until the next newline is comment text.

- *layout\_subset name {layer\_range datatype\_range} ...*

Defines the subset of layout whose extent is required. Multiple layout\_subset statements can be specified, each identified by a unique name that is used to identify this layout subset in the MAP identifier. If MAP is not specified, only one layout\_subset can be used. The layout\_subset defines the subset of the input layout using one or more combinations of layer and datatype numbers. The *layer\_range* can be either a number, a range of numbers, or an asterisk (\*) for all layers. Similarly, *datatype\_range* is also a number, a range of numbers, or

an asterisk (\*) for all data types. The range of numbers is specified using a hyphen between two numbers.

- vboasis\_path *file\_path* [-noCheck]

This specifies the path of the input VB:OASIS file whose extents need to be computed. If an index file exists, then it can be recreated to add the extent information.

The OASIS index file is read or written as *file\_path.fvi*. Thus, the location of the index file is same as that of *file\_path*. This behavior can be changed by specifying a colon-separated list of directories using the UNIX environment variable MDPIndexSearchPath. The directories are searched in the order they are listed for reading or writing the index file.

The -noCheck option can be specified in circumstances when checks on the input VBOASIS layout file are not possible because it has not been generated. This would typically happen when the file is expected to be generated by another SVRF command and thus is not available during parsing for checking purpose. When using -noCheck, extra care should be given to specifying the file path and associated precision values to avoid errors that will be encountered much later.

- vboasis\_precision\_multiplier {*n*[/*d*] | AUTO}

Multiplies the precision by the supplied number *n*, and then scales the data from the file by the same amount to retain the same absolute scale as the original precision. The default value for *n* is 1. The /*d* is used if you wish to express a rational number as a fraction (for example, a multiplier of 1.5x would be declared as “3/2”).

There may be occasions where you might want to reinterpret the precision of the VB:OASIS file specified using the vboasis\_path keyword. For instance, you may want a higher resolution to allow finer shape modifications in embedded SVRF code.

If AUTO is supplied instead of a ratio, Calibre will attempt to infer the correct ratio but will fail if it is not a rational number, or if the combination of inferred numerator and denominator would cause arithmetic overflow. This keyword requires the use of embedded SVRF.

## Description

The MDP OASIS\_EXTENT SVRF operation computes the extent of an OASIS layout. The extent can be calculated for a subset of the layout defined in terms of a combination of layer and datatype numbers called layout\_subset. Each extent is written to the desired output layer as a rectangle.

In the case of multiple layout\_subset parameters, there should always be a matching layer name in the MAP statement.

For all MDP syntax, text must be on lines STRICTLY BETWEEN the left and right brackets (in other words, they cannot be on the same line). The text should not contain a right bracket (]) or left bracket ([).

When using comments and naming the layer name associated with layout\_subset, note that under certain circumstances, the comment characters might be ignored by MDP. In the

## **MDP Oasis\_Extent**

---

following example, the comment start and end are completely ignored and the example runs as if no comments are present.

```
initial_extent = MAGNIFY input_extent BY 2
input_extent = MDP OASIS_EXTENT MAP initial_extent FILE /*
    vboasis_path "tsisram.oas"
    layout_subset initial_extent 1 0
*/
```

This is because nothing that is present on the lines containing right bracket (]) or left bracket ([) is processed by MDP.

### **Example**

```
initial_extent = MAGNIFY input_extent BY 2
input_extent = MDP OASIS_EXTENT MAP initial_extent FILE [
    vboasis_path "tsisram.oas"
    layout_subset initial_extent 1 0
]
```

In this example, “MAP initial\_extent” is optional and unrelated to “initial\_extent = MAGNIFY input\_extent BY 2.” However, if it is used, it should match with “layout\_subset initial\_extent.”

## MDPmerge

Mask data prep operation

```
MDPMERGE format FILE { parameter_block_name | '['
    input_dir dir1
    output_dir dir2
    layout_name name_of_input_layout
    merge_dis distance
    [frame_max_data max_frame_size]
    [common_max_data max_file_size]
    [frame_bde_max_data max_frame_bde_size]
    [common_bde_max_data max_common_bde_size]
    [common_cell_max value]
    [cell_def_max value]
    [cell_loc_max value]
    [frame_width width]
    ']'
}
}
```

### Parameters

For detailed information about the parameters, refer to “[Calibre MDPmerge](#)” in the *Calibre Mask Data Preparation User’s Manual*.

- *format*

**VSB11** is the only value currently supported.

### Description

You can use this operation for the following:

- Optimizing the throughput of the mask writing machine
- Optimizing the parameter control like registration and CD
- Enabling the proximity correction for adjacent placements of individual chips based on certain criteria (for example, the distance from each other, write sequence)

See [Fracture NUFLARE](#) for details on fracturing commands.

## MDPstat

Mask data prep operation

```
MDPSTAT { SMALLOUTSIDETRAP s [e] | SPLITCD w l }
{ INSIDE OF x1 y1 x2 y2 | INSIDE OF LAYER layer1 }
[FILENAME name_of_file] FILE { parameter_block_name | '['
  input_file file
  verify_type {JEOL2DB | VSB2DB | HITACHI2DB | MICRONIC2DB |
VBOASIS2DB}
    [maximum_output_count limit]
    [magnify value]
    [rotate rotate]
    [mirror1 {x | y | x y}]
    [shift x y ]
    '['
    ']'
}
}
```

### Parameters

For detailed information about the parameters, refer to “[Calibre MDPstat](#)” in the *Calibre Mask Data Preparation User’s Manual*.

### Description

Analyzes Hitachi, JEOL, MICRONIC, and NUFLARE (VSB11/VSB12) formatted data based on the following criteria, allowing you to assess the quality of the fracture output:

- Small outside trapezoid detection — detects critically-dimensioned trapezoids whose edges have a specified amount of exposed area.
- Split CD detection — detects critically-dimensioned polygons that have been split lengthwise into trapezoids.

Calibre MDPstat writes statistical data to the Calibre transcript and generates a new layer containing the applicable geometries.

See [Fracture HITACHI](#), [Fracture JEOL](#), [Fracture MICRONIC](#), [Fracture NUFLARE](#), and [Fracture VBOASIS](#) for details on fracturing commands.

## MDPverify

Mask data prep operation

```
MDPVERIFY layer1 [...layerN]
[ {INSIDE OF x1 y1 x2 y2 | INSIDE OF LAYER layer_ext | INSIDE OF EXTENT}]
[FILENAME name_of_file] [MAP mapstring] FILE {parameter_block_name} | '['
    [svrf_layer_name -file1 {layer_name [layer_num [layer_datatype]]}...
     [-file2 {layer_name [layer_num [layer_datatype]]}... | ...
      -directIn {layer_name [layer_num [layer_datatype]]}...]]
[<SVRFSTART>
    svrf_statement1
    ...
    svrf_statementN
<SVRFEND>]
file1 [file2]
verify_type {JEOL2DB | VSB2DB | MEBES2DB | HITACHI2DB |
    JEOL2JEOL | MEBES2MEBES | VSB2VSB | HITACHI2HITACHI |
    MEBES2VSB | MEBES2JEOL | MEBES2HITACHI | MEBES2MICRONIC |
    MICRONIC2DB | MICRONIC2MICRONIC | VSBJOB2VSBJOB |
    JEOL2VSB | JEOL2HITACHI | VSB2HITACHI | VBOASIS2DB |
    VBOASIS2JEOL | VBOASIS2VSB | VBOASIS2VBOASIS |
    VBOASIS2MEBES | VBOASIS2MICRONIC}
[maximum_output_count limit]
[magnify value1 [value2]]
[rotate rotate1 [rotate2]]
[mirror1 {x | y | x y}]
[mirror2 {x | y | x y}]
[shift {x1 y1 | NONE [x2 y2 | NONE]}]
[size size_value]
[reverse_tone {yes | no} [{yes | no}]]
[mask1 layout_name]// applies only to VSBJOB2VSBJOB
[mask2 layout_name]// applies only to VSBJOB2VSBJOB
[field_size x [y]]
[field_size_multiplier m]
[input_extent 1 ix1 iy1 ix2 iy2]
[input_extent 2 ix1 iy1 ix2 iy2]
[vboasis_path filepath [-noCheck]]
[vboasis_precision_multiplier {n/d | AUTO}]
[vboasis_layout_magnify n [d]]
[vboasis_injection [-size bin_size | -size_factor size_multiplier] [-preserve 0 | 1]]
[inject_mask_files]
[oasis_output_path path]
[oasis_output_layers {layer [datatype] } | {{mapstring layer [datatype]}}*}]
[oasis_output_primary [HDB | DIRECTIN | FILE1 | FILE2 | -explicit topcell_name]]
[oasis_output_options [-cblock] [-maximum_vertex {mv | ALL}]]
```

```
[runtime_exception_severity [-file_read_error {continue | stop}]]  
[-file_differences {continue | stop}] [-comparison_region {continue | stop}]  
[-shift_value_overflow {continue | stop}]]  
[temp_directory directory_path]  
[index_mask_files {ALL | NONE | size_in_MB}s}  
'T'  
}
```

## Parameters

For detailed information about the parameters, refer to “[MDPverify](#)” in the *Calibre Mask Data Preparation User’s Manual*.

## Description

Allows comparison of Hitachi, MEBES, JEOL, or NUFLARE (VSB11/VSB12) format files to an original or derived polygon layer, or to other Hitachi, MEBES, JEOL, MICRONIC, or NUFLARE (VSB11/VSB12) files. Performs a Boolean XOR of the inputs and sends the result to the results database.

See [Fracture MEBES](#), [Fracture MEBES](#), [Fracture JEOL](#), [Fracture MICRONIC](#), and [Fracture NUFLARE](#) for details on fracturing commands.

# Merge

Layer operation

**MERGE** *layer* [BY *number*]

## Parameters

- *layer*  
An original layer, layer set, derived polygon layer, or derived edge layer.
- BY *number*  
An optional keyword set, where *number* is a non-positive floating-point number that specifies how much to merge the *layer* polygons.

## Description

For hierarchical Calibre tools, Merge performs a partial flattening of the hierarchy such that the resulting layer is a merged copy of the input layer. For flat Calibre tools and ICVerify Layout, Merge is functionally equivalent to a [Copy](#) operation.

Merge has *limited* uses. Although the result of a Merge operation is still a layer in hierarchical form (versus flat), a considerable amount of partial flattening can occur in creating the layer.

If *number* is omitted, polygon (edge) data is moved up the hierarchy until the entire polygon (edge) is present, but no higher. That is, whole polygons (edges) are generated within a particular cell (an individual polygon or edge is never spread throughout the hierarchy). In hierarchical applications, a layer having these characteristics is often said to be in fully-merged form.

If *number* is present, there is no effect on a derived edge layer. If *number* is 0, data is moved up the hierarchy only to remove all inter-hierarchical overlap of polygons but not necessarily to completely merge whole polygons. In hierarchical applications, a layer having these characteristics is considered in partially-merged form. Negative *number* values cause shapes to cancel out those that overlap them at a strictly higher level; this does not remove inter-hierarchical geometry overlaps caused by overlapping placements. A positive *number* is not used in production rule files.

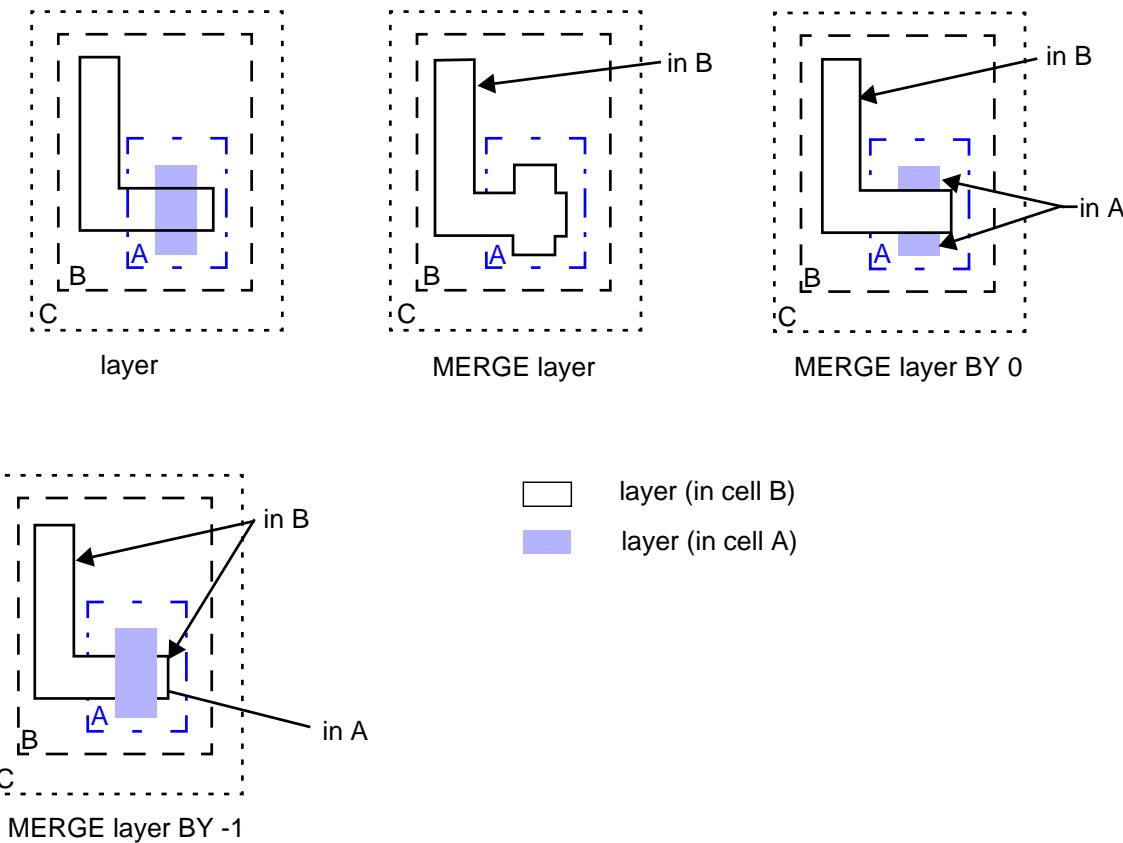
## Merge

### Examples

The Merge operation is shown in Figure 4-173 below where cell A is a sub-cell of B, which is a sub-cell of C. Cell A and B each contain one polygon and these are on the same layer. There are no polygons in C.

MERGE layer merges the polygons and promotes the result to cellB. MERGE layer BY 0 causes the polygon in cell A to be cut into two sections; resulting polygons remain at their original hierarchical levels. MERGE layer BY -1 causes the polygon in cell B to appear in two sections; resulting polygons remain at their original hierarchical levels.

**Figure 4-173. Merge**



# Net

Layer operation

**NET** *layer name* [*name ...*]

## Parameters

- *layer*

A required original layer or layer set, or a derived polygon layer.

- *name*

A required name of a net. You can specify *name* any number of times in one statement. The *name* can be a string variable (see [Variable](#)). A *name* can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters.

The string LVS\_POWER\_NAME may be used as a *name* argument. The string specifies all net names in a [LVS Power Name](#) statement. Similarly, the string LVS\_GROUND\_NAME specifies all net names in a [LVS Ground Name](#) statement. If either of these strings are specified, but the corresponding LVS specification statement is not, then the string is treated as a literal net name.

## Description

Selects all *layer* polygons that belong to the net having the specified *name*. To avoid ambiguity, you must use the parameters in *layer name* order. Net name assignment is discussed under “[Net Name Specification](#)” and “[Label Attachment](#)” in the *Calibre Verification User’s Manual*.

---

### Note



Do not use this operation to obtain net shorting information. The Net operation uses net names assigned by the connectivity extractor. If the connectivity extractor encounters shorted nets, it assumes it has found a single net, and assigns it one name.

---

The connectivity on *layer* must be established through a connectivity operation.

The Net operation uses only top-level net names. The [Text Depth](#) statement controls which labels are used for top-level connectivity text.

By default, net names are case-insensitive; however, the [Layout Preserve Case YES](#) specification statement causes the connectivity extractor to become fully case-sensitive and the Net operation becomes case-sensitive as well.

See also [Not Net](#), [Net Area](#), [Net Interact](#), [Attach](#), [Connect](#), [Label Order](#), [Sconnect](#), and [DRC Incremental Connect Warning](#).

## Examples

```
METAL_SP {
    @ VDD metal must be spaced at 4.5 microns
    @ VCC metal must be spaced at 4 microns
    vdd_metal = metal NET vdd
    vcc_metal = metal NET vcc
    EXT vdd_metal < 4.5
    EXT vcc_metal < 4.0
}
```

## Net Area

Layer operation

### NET AREA *layer constraint*

#### Parameters

- *layer*

A required original layer or layer set or a derived polygon layer.

- *constraint*

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

The constraint must contain non-negative floating-point numbers. It is interpreted in user units squared.

#### Description

Selects all *layer* polygons that belong to a net having a total area on the layer conforming to the *constraint*.

The connectivity on *layer* must be established through a connectivity operation.

See also [Net](#), [Net Area Ratio](#), [Connect](#), [Sconnect](#), and [Stamp](#).

#### Examples

```
connect metal poly by contact
// derive poly shapes greater than 5 um2 on a net
polyx = NET AREA poly > 5
```

## Net Area Ratio

Layer operation

Standard Form:

### NET AREA RATIO

```
{layer1 [SCALE BY value] [COUNT ONLY | PERIMETER ONLY]}
[ {layerN [SCALE BY value] [COUNT ONLY | PERIMETER ONLY]}...]
[OVER]
{d_layer1 [SCALE BY value] [COUNT ONLY | PERIMETER ONLY]}
[ {d_layerN [SCALE BY value] [COUNT ONLY | PERIMETER ONLY]}...]
['[expression']' ] constraint
[INSIDE OF LAYER layer [BY NET]]
[ACCUMULATE [alayer [alayer...]]]
[RDB [ONLY] file_name [MAG value]
 {[BY LAYER] [{rdb_layer [MAXIMUM max_polygon]}...]})]
```

Single-Layer Form:

```
NET AREA RATIO layer1 '[expression']' constraint
[ACCUMULATE [alayer [alayer...]]]
[INSIDE OF LAYER layer [BY NET]]
[RDB [ONLY] file_name [MAG value]
 {[BY LAYER] [{rdb_layer [MAXIMUM max_polygon]}...]})]
```

## Summary

Net Area Ratio selects polygons based on calculations involving nets. By default, the calculation is an area ratio. Net Area Ratio is useful for antenna checks; although it has many other potential uses. It comes in two forms. The [Standard Form](#) is more general. The [Single-Layer Form](#) is used for finding net information about a single layer.

The [Net Area Ratio Accumulate](#) layer operation is closely related to Net Area Ratio and performs ratio accumulation calculations.

To illustrate a common use of the Standard Form, consider this design rule:

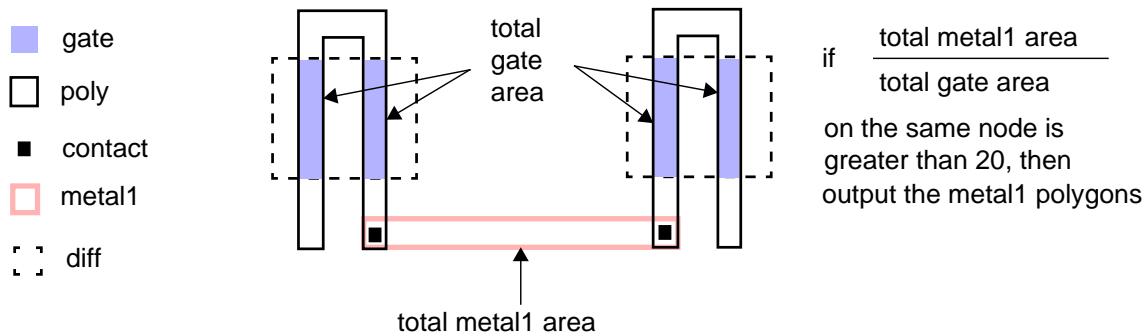
Antenna rule: The ratio of metal1 area to gate area on a net should be no more than 20.

A simple rule check for the design rule could be written this way:

```
gate = poly AND diff
CONNECT poly metal1 BY contact
// level 1 antenna check
ant_1 { NET AREA RATIO metal1 gate > 20 }
```

Figure 4-174 illustrates how the rule check is applied:

**Figure 4-174. NET AREA RATIO metal1 gate > 20**



Further examples begin on [page 1170](#).

## Parameters

- *layer1 ... layerN*

The ***layer1*** parameter is a required original layer, layer set, or a derived polygon layer. The layer must have extracted connectivity information. You can specify *layerN* (which has the same semantics as ***layer1***), along with any associated optional keywords any number of times in one Standard Form operation. These layers contain data for the ratio numerator in the Standard Form.

In the Single-layer Form, only ***layer1*** may be specified.

- **OVER**

An optional keyword that must be included if you specify more than one denominator layer (*d\_layerN*) in a Standard Form operation.

- *d\_layer1 ... d\_layerN*

The ***d\_layer1*** parameter is a required original layer, layer set, or a derived polygon layer. The layer must have extracted connectivity information. You can specify *d\_layerN* (which has the same semantics as ***d\_layer1***), along with any associated optional keywords any number of times in one Standard Form operation. These layers contain data for the ratio denominator in the Standard Form. If more than one *d\_layerN* parameter appears, the **OVER** keyword is required to specify where the denominator layers begin in the parameter string.

- **'[*expression*']**

A numeric expression within square brackets that allows customizable control over the operation computations. The expression defines calculations to be made using the input layers. The value of an ***expression*** is compared to the **constraint**.

The *expression* may involve numbers (including numeric variables), operators, parentheses ( ), and the functions given in [Table 4-30](#):

**Table 4-30. Net Area Ratio Math Functions**

Function	Definition
AREA( <i>input_layer</i> )	Area of <i>input_layer</i> in user units of length squared.
COUNT( <i>input_layer</i> )	Count of polygons on the <i>input_layer</i> .
PERIMETER( <i>input_layer</i> )	Total perimeter of polygons on the <i>input_layer</i> in user units.
SQRT(x)	square root of x
EXP(x)	exponential (base e) of x
LOG(x)	natural logarithm of x
SIN(x)	sine of x in radians
COS(x)	cosine of x in radians
TAN(x)	tangent of x in radians
MIN( <i>expression1,expression2</i> )	Returns the lesser value of the two expressions. The expressions must be of the same form as the primary Net Area Ratio <i>expression</i> , without the square brackets.
MAX( <i>expression1,expression2</i> )	Returns the maximum value of the two expressions. The expressions must be of the same form as the primary Net Area Ratio <i>expression</i> , without the square brackets.

In [Table 4-30](#), x is a numeric argument (including numeric variables and other numeric-valued expressions). There is no compile-time or runtime exception checking on the arguments of these functions, and unreasonable or undefined values may result from certain arguments. The *input\_layer* is an input layer in the same Net Area Ratio statement and is not an ACCUMULATE layer, otherwise a compilation error results. Here is an example expression:

```
[ (area(ipoly) + area(metal1)) / area(gate) ]
```

The expression must not result in strictly negative values because such values cannot be checked by a *constraint*. Division by zero does *not* satisfy any *constraint*.

Parentheses have the highest precedence for the calculation of numeric values and may be used for grouping of terms.

**Operators** — Unary operators (+, -, !, ~) require only one numerical argument and all such operators have the same precedence. The + and - operators are the usual positive and negative signs. The other unary operators do the following:

**! operator** — Returns 0 (or false) if its argument is non-zero and 1 (or true) if its argument is 0.

**~ operator** — Returns 0 (or false) if the argument is positive and 1 (or true) if the argument is non-positive.

[Table 4-3](#) shows some useful combinations of these operators.

The binary operators ^, \*, /, +, - require two numerical arguments. The ^ operator is the same as the C language pow( ) function, where  $x^y$  means  $x$  raised to the  $y$  power. The \*, /, +, and - operators are multiplication, division, addition, and subtraction, respectively, and have the customary precedence. The ^ operator has the same precedence as \* and /.

The binary operators || (OR) and && (AND) are the same as the C language operators of the same type, except that they have the same precedence as binary + and -. They expect numeric-valued inputs. For example:

AREA(via) && AREA(met) returns 1 if both inputs are non-zero and 0 otherwise.

AREA(via) || AREA(met) returns 1 if either input is non-zero and 0 otherwise.

The **expression** supersedes the SCALE BY, PERIMETER ONLY, COUNT ONLY, and to a large extent, the OVER keywords in the [Standard Form](#). The **expression** calculations are in user units.

The **expression** is mandatory in the [Single-Layer Form](#).

For more details, see “[Expressions](#)” on page 1162.

- **constraint**

A required string that is one of the constraints listed in the “Constraint Notation” or “Alternate Notation” column of [Table 2-2](#) on page 45. It is interpreted as the constraining interval of values of the operation.

The tolerance for comparing the **constraint** to the value computed by the operation is controlled by the [DRC Tolerance Factor NAR](#) statement. The default tolerance is 0.005.

- **INSIDE OF LAYER layer [BY NET]**

An optional keyword and layer name set that causes the Net Area Ratio calculation for the input *layerN* polygons to be taken only for the portions of the *layerN* polygons that are inside polygons on the INSIDE OF LAYER *layer* parameter. The *layer* parameter must be an original or derived polygon layer. This keyword set may not be used with the RDB keyword set.

If BY NET is specified, then the ratio calculation is performed with respect to polygons on the input layers that belong to the same electrical node. Connectivity information must be present on the INSIDE OF LAYER *layer* parameter in this case.

INSIDE OF LAYER may be specified with ACCUMULATE, but not ACCUMULATE *alayer*.

For complete details about INSIDE OF LAYER behavior, see “[INSIDE OF LAYER](#)” on page 1159.

- ACCUMULATE [*alayer* [*alayer...*]]

An optional keyword set. ACCUMULATE specifies that Net Area Ratio values are attached to the derived output layer, called a Net Area Ratio accumulation (NARAC) layer.

The optional *alayer* is a derived layer from a previous Net Area Ratio operation with ACCUMULATE specified, or a [Net Area Ratio Accumulate](#) operation, which specifies to add NARAC layer values from *alayer* to a new output layer. The *alayer* and *d\_layer1* must have the same layer of origin. The output layer can either be a derived layer or an error layer.

If more than one *alayer* parameter is specified, then the accumulation ratio values for each specified layer appear in the RDB. The RDB keyword is required when there is more than one *alayer* parameter. If RDB is not specified in this case, a compiler error results. There is no change to the accumulation calculations when more than one *alayer* is specified; the accumulation values are simply applied to the additional *alayer* parameters.

The *alayer* parameter may not be specified with INSIDE OF LAYER.

For details about ACCUMULATE behavior, see “[Ratio Accumulation](#)” on page 1160.

For information about ACCUMULATE RDB format, see “[Results Database \(RDB\) Creation](#)” on page 1164.

- RDB [ONLY] *filename* [MAG *value*]

An optional keyword set, where *filename* is an ASCII results database (RDB) with detailed statistics (area, perimeter, and so forth) arranged by net id. Subdirectories in a pathname are created as required.

This RDB database is *in addition to* the usual nmDRC Results Database, and can be loaded into Calibre RVE or Pyxis Layout. This keyword may not be specified with an ==0 *constraint*.

RDB *filename* — Specifies RDB output to the *filename*. If specified without the ACCUMULATE keyword, statistics come from the first numerator layer. If specified with the ACCUMULATE keyword, statistics come from the first denominator layer. Additionally, the *ldb\_layer* parameter can specify polygon data to appear in the RDB database.

RDB ONLY *filename* — Specifies results are sent to the RDB database only. No geometric data is sent to the usual nmDRC results database. If specified without the ACCUMULATE keyword, polygons come from the first numerator layer. If specified with the ACCUMULATE keyword, polygons come from the first denominator layer. Additionally, the *ldb\_layer* parameter can specify polygon data to appear in the RDB database.

**MAG *value*** — Specifies the results sent to the RDB *filename* are magnified by the specified *value*. The *value* must be a positive floating point number, which is dimensionless. When specified, the MAG keyword overrides a [DRC Magnify NAR](#) statement.

The *rdb\_layer* parameter is recommended for most cases when an RDB is requested. If the *rdb\_layer* parameter is not specified, then no geometry is output to the RDB, just rule check statistics.

The RDB keyword set is required if ACCUMULATE is specified with more than one *alayer* parameter.

The RDB keyword set may not be specified with INSIDE OF LAYER.

For information about Net Area Ratio RDB format, see “[Results Database \(RDB\) Creation](#)” on page 1164.

- **BY LAYER**

An optional keyword set that instructs Calibre to generate a result, similar to a rule check result, for each (output net, *rdb\_layer*) coordinate. You must specify at least one *rdb\_layer* if you use the BY LAYER keyword, otherwise the tool issues a compilation error. The rule check results contain statistics and polygons for each *rdb\_layer*. You should use this keyword whenever generating RDB results.

- ***rdb\_layer* [MAXIMUM *max\_polygon*]**

An optional keyword set that instructs Calibre to generate detailed statistics and polygons for the specified *rdb\_layer*. This keyword set can be specified any number of times in one statement and must be specified with the RDB keyword.

*rdb\_layer* — An original layer, layer set, or a derived polygon layer. This layer must appear as a ***layerN*** or ***d\_layerN*** within the same Net Area Ratio operation.

**MAXIMUM *max\_polygon*** — Optional keyword and non-negative integer that specify the maximum number of polygons per net the tool outputs for the respective RDB layer. Calibre outputs all *rdb\_layer* polygons of a net if you do not specify a MAXIMUM *max\_polygon* set.

The *rdb\_layer* parameter is recommended for most cases when an RDB is requested. If the *rdb\_layer* parameter is not specified, then no geometry is output to the RDB, just rule check statistics.

- **SCALE BY *value*, COUNT ONLY, and PERIMETER ONLY**

These keywords are provided for backward compatibility. They have largely been replaced by the *expression*. Net Area Ratio uses *database units* for area and perimeter measurements, if you do not use an *expression*. This implies the **Precision** is multiplied by user units of length to calculate areas and perimeters for Net Area Ratio operations not using an *expression*.

**SCALE BY *value*** — An optional keyword set, where *value* is a non-zero floating-point number that instructs the tool to multiply the nodal area, perimeter, or count of the layer

(*layerN* or *d\_layerN*) by *value* prior to its use in the calculation. This is often used with COUNT ONLY and PERIMETER ONLY to get the units of measurement (area, length, and polygon count) in a ratio to match.

COUNT ONLY — An optional keyword that instructs the tool to use the nodal polygon count of the layer in the calculation of the net ratio instead of the area, for the associated layer. COUNT has no units of measurement, so if you want to have COUNT related to perimeter (units of length) or area (units of length squared), use SCALE BY and a factor that uses your [Precision](#) (for perimeter) or Precision squared (for area).

PERIMETER ONLY — An optional keyword that instructs the tool to use the nodal perimeter in the calculation of the net ratio instead of the area, for the associated layer. Internal calculations of length are done in database units, so you may need to consider using a SCALE BY factor to relate perimeter to area to get the units of a ratio to match. Using the Precision value as part of the factor to multiply perimeter by obtains the correct dimensions of area (length squared) in database units.

For example, suppose you want to ensure the ratio of sidewall area of poly to the area of gate is no more than 50. Suppose that poly is 0.10 um thick. Here is a possible rule check:

```
area_check {NET AREA RATIO poly PERIMETER ONLY SCALE BY 100  
gate > 50}
```

The sidewall area of poly is the perimeter of poly multiplied by the thickness. PERIMETER ONLY finds the perimeter of poly shapes. The SCALE BY 100 accounts for the thickness of poly (.10 um, in user units) and ensures the result is in database units. Assuming Precision is 1000,  $0.10 * 1000$  is 100. This is the SCALE BY factor in the rule check. The dimensions in the ratio are then all in terms of area in database units.

Using an expression, a similar rule check would be:

```
area_check {NET AREA RATIO poly gate > 50  
[ PERIMETER(poly) * .10 / AREA(gate) ]}
```

Note that the *expression* uses user units, not database units, so Precision is not a concern.

## Description

Net Area Ratio has two syntax types (see [page 1152](#)): Standard Form and Single-layer Form. The [Net Area Ratio Accumulate](#) operation is treated separately from Net Area Ratio, but is closely related.

## Standard Form

The Standard Form operation selects all polygons from *layer1* that lie on an electrical node, such that the ratio of the total area of *layerN* polygons on that node to the total area of *d\_layerN* polygons satisfies the given *constraint*. The *layerN* (including N=1) parameters are treated as numerator layers and the *d\_layerN* (including N=1) parameters are treated as denominator layers in the default area ratio calculation when used in the default configuration.

The term “area” is used loosely here, as the calculated ratio can pertain to other quantities like polygon counts, perimeters, or custom *expression* values. The unit dimensions of the ratio that

is calculated are determined by the *expression*, if provided; otherwise, the default area ratio is dimensionless.

Note that all *layer1* polygons on the node are selected if the ratio meets the *constraint*. The behavior is different if you use the ACCUMULATE, *rdb\_layer*, or BY LAYER parameters. For ACCUMULATE, *d\_layer1* polygons are selected. For *rdb\_layer* (and BY LAYER), you can specify input layers other than *layer1* or *d\_layer1*, which can be selected for output.

Here is a more precise definition of the basic Standard Form operation. Consider, this example:

**NET AREA RATIO L1 L2 ... Ln OVER D1 D2 ... Dm constraint**

$L_i$  are *numerator layers* and  $D_i$  are *denominator layers*. For each net N containing  $L_1$ , let X be the total area (or count, or perimeter) of all polygons in  $L_i$  on net N, and let Y be the total area (or count, or perimeter) of all polygons in the denominator layers on net N. Polygons on  $L_1$  for net N are output if the following hold:

1. Y is not equal to 0, and the ratio X / Y meets the given *constraint*.
2. Y is 0, n is 1, and the constraint is exactly specified as == 0. This is a special case that allows the ratio denominator to be 0. This case is often used when attempting to select polygons on nets that do not have certain layers (for example, diode). Here is an example:

```
// Select metall1 polygons on nets without diodes:  
y = net area ratio metall1 diode == 0
```

This can be used to exclude nets protected by diodes from antenna ratio calculations.

## INSIDE OF LAYER

This keyword set restricts the Net Area Ratio calculation to take place inside of a specified layer. The INSIDE OF LAYER parameter *layer* is not required to have connectivity unless the BY NET keyword is used. If INSIDE OF LAYER is specified, then the operation is a node-preserving layer constructor (rather than a selector, which is the default).

INSIDE OF LAYER may be specified with ACCUMULATE, but not ACCUMULATE *alayer*.

The Net Area Ratio operation with INSIDE OF LAYER specified as follows:

**Z = NET AREA RATIO L<sub>1</sub> ... L<sub>n</sub> [expression] constraint INSIDE OF LAYER I [ACCUMULATE]**

uses the following calculation method.

For each polygon P on layer I:

- a. Set  $X_i = L_i \text{ AND } P$ , and passing connectivity from  $L_i$  to  $X_i$ . If ACCUMULATE is specified, the *Inside* operation is used instead of AND.
- b. Perform the operation:  

$$Z_P = \text{NET AREA RATIO } X_1 X_2 \dots X_n \text{ expression constraint [ACCUMULATE]}$$
- c. Add  $Z_P$  to Z.

## Net Area Ratio

---

In other words, the intersection of each of the input *layerN* parameters with each polygon from the INSIDE OF LAYER parameter is taken. The output layers of this step have the connectivity of the respective input *layerN*. Then, the Net Area Ratio operation is performed as though these output layers were the input layers. The aggregate output of this revised Net Area Ratio operation is the output of the original operation.

The INSIDE OF LAYER parameter is valuable in many applications. For example, consider the following rule:

Every polygon on layer A must be connected to some polygon on layer B inside of the same polygon on layer C.

With INSIDE OF LAYER it is as follows:

```
rule { NET AREA RATIO A B == 0 INSIDE OF LAYER C }
```

Use of the BY NET keyword with INSIDE OF LAYER requires that connectivity be successfully established on the INSIDE OF LAYER *layer* parameter. If BY NET is specified as follows:

**Z = NET AREA RATIO L<sub>1</sub> ... L<sub>n</sub> [expression] constraint INSIDE OF LAYER I BY NET [ACCUMULATE]**

then this is the calculation method.

For each net N on layer I:

- a. Set  $I_N$  be the set of all polygons on layer I on net N.
- b. Let  $X_i = L_i \text{ AND } I_N$ , and passing connectivity from  $L_i$  to  $X_i$ . If ACCUMULATE is specified, the **Inside** operation is used instead of AND.
- c. Perform the operation:  
 $Z_N = \text{NET AREA RATIO } X_1 \dots X_n \text{ expression constraint [ACCUMULATE]}$
- d. Add  $Z_N$  to Z.

In other words, each net for the INSIDE OF LAYER parameter is determined. The set of polygons that belong to each net is then determined. For each of these sets of polygons, their intersection with the input layers  $L_n$  is taken. The output layers of this step have the connectivity of the input layers. Then, the Net Area Ratio operation is performed as though these output layers were the input layers. The aggregate output of this revised Net Area Ratio operation is the output of the original operation.

## Ratio Accumulation

For antenna checks in some processes, the antenna ratios checked at each level of connectivity may not accurately model accumulated charge. For example, the antenna ratio for a net may be acceptable for the first- and second-level antenna checks. However, if added together, the cumulative value may be sufficiently large to imply destruction of the gates through accumulation of antenna charge.

This enhanced antenna effect can be checked by accumulating the antenna ratios for the individual gates at each level of connectivity. That is, the antenna ratio for each shape in the

first denominator layer of a Net Area Ratio operation is saved. Subsequent Net Area Ratio operations then accumulate these saved ratios. The Net Area Ratio operation specified with ACCUMULATE performs this check as follows:

1. If *alayer* is not specified, the Net Area Ratio operation creates a NARAC (net area ratio accumulation) layer consisting of all polygons in *d\_layer1* on net N where the ratio X/Y meets the given *constraint*, as described previously. The actual ratios are attached to the *d\_layer1* shapes.
2. If *alayer* is specified, it must be the output from a Net Area Ratio operation specified with ACCUMULATE, or a [Net Area Ratio Accumulate](#) operation. The computation proceeds as in Step 1. However, if a polygon in *d\_layer1* is in *alayer*, then the saved ratio in *alayer* is first added to the ratio X/Y for that polygon, and this new ratio is then tested against the *constraint*. If the ratio satisfies the *constraint*, the ratio is attached to the output layer.

The *d\_layer1* and *alayer* must have the same layer of origin (see “Layer of Origin” in the [Calibre Verification User’s Manual](#)).

If more than one *alayer* is used, the second and subsequent layers are only used in RDB output. They do not affect the computations. If more than one *alayer* is specified, an RDB specification is required. The NARAC ratios are reported in the RDB for every *alayer*.

The Net Area Ratio operation with ACCUMULATE specified are used as layer selectors for this case only; otherwise, they are layer constructors. For hierarchical applications, all NARAC layers are treated hierarchically.

**Example:**

```
/* Level one antenna operation. Assume proper connectivity extraction and
layer derivations. */

AC1 = NET AREA RATIO m1 gate >= 0 ACCUMULATE RDB acl.rdb m1 gate BY LAYER
```

AC1 contains all GATE polygons, with the M1-to-GATE area ratio information attached to the GATE polygons. Later, you can do this:

```
// Level two antenna operation. Add results to level one.

AC2 = NET AREA RATIO m2 gate >= 0 ACCUMULATE AC1 RDB ac2.rdb m2 gate
BY LAYER
```

AC2 contains all GATE polygons with the M2-to-GATE area ratio included. The ratio information from AC1 is added to the M2-to-GATE area ratio information. The cumulative ratios are attached to the GATE polygons. In this way, cumulative charge effects are stored in AC2 for the GATE layer. You could also send results to the results database this way:

```
//Level 2 antenna check.
ant.2 {NET AREA RATIO M2 GATE > 100 ACCUMULATE AC1}
```

Note that a constraint of  $\geq 0$  would probably not be used here because the intent is to find violations, not simply to accumulate ratio information.

If you specify ACCUMULATE *alayer* with the RDB keyword, and you specify the **d\_layer1** layer as an *rdb\_layer*, then any geometry from the **d\_layer1** layer that appears as a rule check result in the RDB has an attached property:

`AR value`

where AR means accumulated ratio and *value* is the floating-point, accumulated ratio of the individual polygon. It is this property value that shows the accumulated value for a polygon that fails the check. This property value is used to compare against the **constraint**.

For each *alayer* specified, the accumulated ratios are written to the RDB using this format:

`AR alayer value`

If an *alayer* has no object intersecting the **d\_layer1** layer, then the following is written to the RDB:

`AR alayer NP`

where NP means not present.

Following are the differences between the Net Area Ratio operation specified with and without ACCUMULATE:

- Without ACCUMULATE: output is from **layer1** (first numerator layer).  
With ACCUMULATE: output is from **d\_layer1** (first denominator layer).
- Without ACCUMULATE: acts as a layer selector (from **layer1**), hence, node-preserving.  
With ACCUMULATE: acts as a non-node-preserving layer constructor, unless more than one accumulation layer is specified.
- Without ACCUMULATE: the == 0 constraint is a special case where the denominator of the ratio is allowed to be 0. See [page 1159](#).  
With ACCUMULATE: == 0 constraint is not a special case.

## Expressions

Expressions define calculations to be made using the input layers. These calculations can be area ratios, but do not have to be. These calculations can make the Net Area Ratio operation an advanced net selection tool.

By default, Net Area Ratio calculates the area ratios of numerator layers to denominator layers on a given net. If you specify an *expression*, the evaluation rules for the Net Area Ratio statement change as follows:

- The *expression* determines the value to compare to the **constraint**. Values that satisfy the **constraint** are output. Division by zero in an *expression* satisfies no **constraint**.
- The presence of an *expression* makes specification of PERIMETER ONLY, COUNT ONLY, and SCALE BY keywords invalid. All of these keywords may be effectively implemented in the *expression* itself.

- The concept of numerator versus denominator layers can be specified in the *expression*. The OVER keyword is not needed, but can enhance readability for some operations.
- A Net Area Ratio *expression* is always evaluated in user units. There is no need to use scale factors in order to convert from database units.

The evaluation rule differences for a Net Area Ratio operation with an *expression* are described as follows:

- For the non-accumulated Net Area Ratio operation in Standard Form:

**NET AREA RATIO L<sub>1</sub> L<sub>2</sub> ... L<sub>n</sub> OVER D<sub>1</sub> D<sub>2</sub> ... D<sub>m</sub> [*expression*] *constraint***

For each net N containing L<sub>1</sub>, the *expression* is evaluated, where each layer function (AREA, PERIMETER, and COUNT) present in the *expression* returns the corresponding value of its input layer on net N. Polygons on net N in L<sub>1</sub> are output if evaluation of the *expression* did not result in division by zero, and the value of the *expression* meets the given ***constraint***. There is no == 0 constraint special case (see [page 1159](#)) if an *expression* is present.

- For the Accumulated Net Area Ratio operation in Standard Form:

**NET AREA RATIO L<sub>1</sub> L<sub>2</sub> ... L<sub>n</sub> OVER D<sub>1</sub> D<sub>2</sub> ... D<sub>m</sub> [*expression*] *constraint*  
ACCUMULATE [*alayer* ...]**

For each net N containing D<sub>1</sub>, the *expression* is evaluated, where each layer function (AREA, PERIMETER, and COUNT) present in the *expression* returns the corresponding value of its input layer on net N. The operation creates a NARAC layer consisting of all shapes in D<sub>1</sub> on all nets N such that evaluation of the *expression* did not result in division by 0, and the value of the *expression* meets the given ***constraint***. The value of the *expression* itself is attached to the D<sub>1</sub> shapes.

When *alayer* is used, and a polygon in D<sub>1</sub> is in *alayer*, and evaluation of the expression did not result in division by 0, the saved value of the *alayer* is added to the value of the *expression*. This new value is tested against the ***constraint*** and potentially attached to shapes in the output layer.

Here is an example of a layer derivation with an *expression*:

```
b = NET AREA RATIO M2M M1M PGM gate > 100
    [ (AREA(M2M) + (.1 * PERIMETER(M1M)) + AREA(PGM)) / AREA(gate) ]
```

The expression value is evaluated to see if it meets the > 100 constraint.

The complete list of functions and operators is shown in the Parameters section. The following are notes on two of the operators:

**! operator** — Returns “0” (or false) if its argument is non-zero and “1” (or true) if its argument is zero. Used in front of one of the functions AREA(), PERIMETER(), or COUNT() (especially AREA())—the most efficient of the three), the ! operator can be valuable in performance optimizations for antenna checking where regions connected to certain components, like diodes, are not to be counted in antenna area calculations. Instead of first excluding these regions, the exclusion can be placed in the primary check itself.

For example, suppose you want to check only the nets that are not diode-protected. If a diode is present, !AREA(diode) returns a “0” because AREA(diode) is non-zero. This can be used in an *expression* to eliminate nets with diode protection:

```
//If diodes are present, !AREA(diode) is 0.  
rule {NET AREA RATIO metall poly gate diode > 100  
      [((AREA(metall) + AREA(poly)) * !AREA(diode)) / AREA(gate)]  
}
```

The former rule check may be preferred instead of this one:

```
X = NET AREA RATIO metall diode == 0  
rule {NET AREA RATIO X poly gate > 100}
```

You can use successive ! operators, for example: !!AREA(gate). If AREA(gate) is not 0, !AREA(gate) returns “0”. Therefore, !!AREA(gate) returns a “1”. In effect, !!AREA(gate) acts as a detector of whether or not gate polygons are present on a net. It returns “1” (or true) if gates are present.

**~ operator** — Returns “0” (or false) if the argument is positive and “1” (or true) if the argument is non-positive. This is different from the ! operator. Recall, the ! operator returns “0” if the argument is positive or negative and returns “1” if the argument is zero. The ~ operator is particularly useful where the sign of the argument is of concern.

[Table 4-3](#) shows some useful combinations of the ! and ~ operators.

## Results Database (RDB) Creation

Net Area Ratio has several results database (RDB) output options. These are shown in [Table 4-31](#).

**Table 4-31. Database Output for Net Area Ratio**

RDB keyword set	Output databases
RDB not specified	nmDRC Results Database
RDB specified	nmDRC Results Database and RDB database
RDB ONLY specified	nmDRC Results Database (with no polygon information from the Net Area Ratio operation) and RDB database

The RDB results database, created through the use of the RDB [ONLY] keyword set, contains numeric statistics based on each net. The database can be loaded into Calibre RVE or Pyxis Layout. The RDB results database is different from (but similar to) the standard nmDRC Results Database that is always created for nmDRC operations. For information on the structure of the standard nmDRC results database, refer to the section “ASCII DRC Results Database Format” in the [Calibre Verification User’s Manual](#).

Each net that generates output from the Net Area Ratio operation generates a rule-check-style result within the RDB database. The following describes the format of the results. All sections

of the RDB database that are the same as the usual nmDRC results database have the standard nmDRC results database format.

- **Rule-check-style result name** — The name of the result is:

*output-layer-name\_number[\_input-layer-name]*

where *output-layer-name* is the name of the layer created by the operation, *number* is an internally generated sequential integer that aids in distinguishing different output nets for the same operation. The *input-layer-name* is used only if you specify BY LAYER and is the name of the respective *rdb\_layer*.

- **Check Text Header** — This header is used to display statistics computed for the net.

The first line of the check text will be the text of the Net Area Ratio operation, excluding the RDB keyword and parameters.

The second line of check text contains cell, net, and value information in the following form:

```
CELL=cell_name FPC=fpc_number NET=net_number NETNAME=net_name
VALUE=value_number
```

where each field is separated by one blank space, and these are definitions:

- CELL=*cell\_name* — The name of the cell that completely contains the output net. However, it may not be the lowest such containing cell. For flat Calibre nmDRC and ICrules this is always the top-level cell name.
- FPC=*fpc\_number* — A positive integer that gives the number of placements of this cell from the flat view. For flat Calibre nmDRC and ICrules this number is always 1. For hierarchical Calibre nmDRC, you can use this number to determine the number of output nets corresponding to this rule check in the flat view.
- NET=*net\_number* — A positive integer representing an internal node number.
- NETNAME=*net\_name* — The name of the net when *cell\_name* is the top-level cell and the net name for the output net is available. If **DRC Cell Text YES** is specified, and a net name is available in a lower-level cell, then the *net\_name* appears for that cell. Otherwise, *net\_name* is empty.
- VALUE=*value\_number* — A floating-point number that represents the actual value computed for this output net (not polygon). This is the value tested against the **constraint** parameter for a given Net Area Ratio operation unless ACCUMULATE is specified (in which case the AR property value is tested). When you specify an *expression* parameter, *value\_number* is the value of the expression, otherwise *value\_number* is the area ratio of numerator to denominator layers.

- ***rdb\_layer* Statistics** — These statistics appear when you specify an *rdb\_layer* parameter.

When you specify one or more *rdb\_layer* parameters, the third line of check text displays statistics on the specified layers in the following form:

```
layer-name AREA=value PERIMETER=value COUNT=value DBCOUNT=value
```

where each field is separated by one blank space, and the following definitions hold:

- *layer-name* — The name of a specified *rdb\_layer*.
- AREA=*value* — A floating-point number in user units, specifying the area of the given layer on the output net. The tool reports *value* only if it was computed in the course of the operation; otherwise, only AREA= is printed.
- PERIMETER=*value* — A floating-point number in user units, specifying the perimeter of the given layer on the output net. The tool reports *value* only if it was computed in the course of the operation; otherwise only PERIMETER= is printed.
- COUNT=*value* — A floating-point number in user units, specifying the count of polygons on the given layer on the output net. The tool reports *value* only if it was computed in the course of the operation; otherwise only COUNT= is printed.
- DBCOUNT=*value* — A non-negative integer specifying the number of polygons on the *rdb\_layer* on the output net.

The following example shows a Net Area Ratio statement in a derived layer:

```
b = NET AREA RATIO M2M M1M PGM gate > 100
  [ (AREA(M2M)+PERIMETER(M1M)+AREA(PGM))/AREA(gate) ]
  RDB output.file M2M M1M PGM gate
```

Here is the corresponding results output:

```
...
b_240
5 5 6 May 4 09:18:19 2000
b = NAR M2M M1M PGM gate > 100
  [ (AREA(M2M)+PERIMETER(M1M)+AREA(PGM))/AREA(gate) ]
CELL=SARY FPC=15 NET=1215 NETNAME= VALUE=619.094444444
M2M AREA=71.44 PERIMETER= COUNT= DBCOUNT=1
M1M AREA= PERIMETER=345.89 COUNT= DBCOUNT=2
PGM AREA=19.08 PERIMETER= COUNT= DBCOUNT=1
gate AREA=3.6 PERIMETER= COUNT= DBCOUNT=1
<polygon data>
...
```

Polygons for each layer appear as coordinate data (DRC results) in the RDB results database according to standard results database form. The maximum number of polygons to output for each layer may be specified by the *max\_polygon* parameter following the optional MAXIMUM keyword for an *rdb\_layer*. This may be any non-negative integer (including 0). If not specified, all polygons of the given layer on the output net are put into the RDB results database.

The coordinate data is in top-cell space only. In Calibre nmDRC-H, if the containing cell is not the top-level cell, then one placement of that cell (in the flat viewpoint) is chosen, and the polygonal data is transformed into world coordinates. This is the identical algorithm used for standard DRC results.

The polygons are output in order of the layers' appearance following the RDB keyword. The number of polygons output for a layer is calculated, as previously discussed, by the “DBCOUNT=” value in the checktext line for the layer. The total number of polygons output for a net is part of the standard nmDRC results database format, and appears on the data line following the nmDRC rule check name corresponding to the net.

Polygons are segmented at the default value of 4096 vertices. There is no warning issued when segmentation occurs.

The optional BY LAYER keyword specifies that, instead of generating a nmDRC rule check for each output net, a nmDRC rule check should be generated for each (*output-net, rdb\_layer*) pairing. This keyword is recommended for all Net Area Ratio operations where RDB is specified.

For each nmDRC rule check having BY LAYER specified, only the statistics and polygons for that layer are included in the checktext and geometric output for the rule check. However, the first lines of checktext (operation text, cell name, net, net name, value), which contain net-wide information, are always present, and are duplicated in each appropriate nmDRC rule. The name of the nmDRC rule check for a specific layer, is this:

```
output-layer-name_number_rdb_layer
```

Here is an example of a derived layer using the BY LAYER keyword:

```
b = NET AREA RATIO M2M M1M PGM GATE > 100
[ (AREA(M2M)+PERIMETER(M1M)+AREA(PGM)) / AREA(GATE) ]
RDB rdb.db M2M M1M PGM GATE BY LAYER
```

This yields the following results output:

```
...
b_240_M2M
1 1 3 May 4 09:18:19 2000
b = NAR M2M M1M PGM GATE > 100
[ (AREA(M2M)+PERIMETER(M1M)+AREA(PGM)) / AREA(GATE) ]
CELL=SARY FPC=15 NET=1215 NETNAME= VALUE=619.094444444
M2M AREA=71.44 PERIMETER= COUNT= DBCOUNT=1
p 1 8
<polygon data>
...
```

The output polygon data is organized by layer.

BY LAYER is intended as a convenience to view the coordinate data for multiple layers, but only one layer at a time. It is possible to parse the RDB results database created without using BY LAYER and convert it to one similar to that created using BY LAYER. The net number appearing in each nmDRC rule check is internal. It is guaranteed to be consistent within the same Net Area Ratio operation execution, or within the same set of concurrently-executed

## Net Area Ratio

---

Net Area Ratio operations. It is not guaranteed to be consistent, however, across multiple, non-concurrent Net Area Ratio operations.

If ACCUMULATE is specified, but with no *alayer* parameter, then RDB semantics are identical to those if ACCUMULATE is not specified, except that an output net is defined as a net with geometry in the first denominator layer having a value that meets the operation's **constraint**.

If ACCUMULATE *alayer* and RDB are specified, then a net is output to the RDB if at least one polygon from the **d\_layer1** layer on the net meets the operation's **constraint**. If you specify the **d\_layer1** layer as an *rdb\_layer*, then any geometry from that layer that appears in the ensuing RDB as a rule check result will have this attached property: AR *value*. The value is the floating-point accumulated ratio of the individual polygon that fails the rule check.

For each *alayer* specified, the accumulation ratio is reported with this attached property: AR *alayer value*. If the *alayer* has no object intersecting the **d\_layer1** layer, then this is reported: AR *alayer NP*, where NP means not present.

Although a net is considered RDB output if the accumulated ratio on at least one polygon in the first denominator layer and from the net meets the operation's constraint, only those polygons on the first denominator layer whose individual ratio actually satisfies the operation's **constraint** are output.

Here is an example of RDB output when ACCUMULATE is specified with multiple *alayer* parameters. For this rule check:

```
BAD_GATE {  
    NET AREA RATIO M3 GATE > 100  
    ACCUMULATE ACC_M2 ACC_M1 ACC_PO  
    RDB M3.rdb GATE  
}
```

This might be the results data:

```
p 6 4  
AR 48.823  
AR ACC_M2 24.05407  
AR ACC_M1 14.63415  
AR ACC_PO 10.13478  
443530 1230825  
443630 1230825  
443630 1231265  
443530 1231265
```

Here the AR properties indicate the history of the charge buildup on the antenna.

RDB results databases may be in common among multiple Net Area Ratio RDB operations. The first open of an RDB file in a nmDRC run truncates the file and writes the header line consisting of top cell name database precision as specified by the [DRC Results Database Precision](#) statement. Subsequent file opens are in append mode. If an RDB file cannot be opened, then a warning is issued and the operation continues as if RDB were not specified. The RDB name is checked for collisions with ASCII [DRC Results Database](#) or [DRC Check Map](#) database names.

A conflict generates a compiler error. The names of RDBs generated by other layer operations are not checked.

The geometric results in the RDB can be magnified by using either the MAG keyword in the operation itself, or through the [DRC Magnify NAR](#) specification statement. If both are used, the MAG keyword takes precedence.

Finally, RDB is supported only in Calibre nmDRC, nmLVS (ERC), and ICrules; warnings are issued when PEX applications attempt to execute a Net Area Ratio RDB operation, and the operation continues as if RDB were not specified.

## Single-Layer Form

Single-layer Form is useful when you want net-based calculations for a single layer only. An ***expression*** is required in the Single-layer Form. The input ***layer1*** behaves like ***layer1*** in the Standard Form (ACCUMULATE not specified) or like ***d\_layer1*** in the Standard Form (ACCUMULATE is specified). There is no == 0 constraint special case (see [page 1159](#)) in the Single-layer Form. For example:

```
// Derive a NARAC layer where each gate polygon has its
// individual area attached:
CONNECT gate
area_gate = NET AREA RATIO gate >= 0 [ AREA( gate ) ] ACCUMULATE
```

The Single-layer form supports the BY NET keyword, which the Standard Form does not. See [“INSIDE OF LAYER”](#) on page 1159 for details.

Other than the aforementioned differences, the Single-layer form behaves like the Standard Form.

## Requirements and Restrictions

- The connectivity on the input layers must be established.
- The RDB keyword may not be specified with the INSIDE OF LAYER keyword.
- The ***d\_layer1*** and ***alayer*** must be geometrically identical.
- The ***alayer*** parameter may not be specified with the INSIDE OF LAYER keyword.
- The RDB keyword is required if more than one ***alayer*** parameter is specified.
- The BY NET keyword can only be specified in the Single-layer form with the INSIDE OF LAYER keyword. Connectivity is required on the INSIDE OF LAYER parameter in this case.
- In hierarchical applications, COUNT ONLY can be inefficient if applied to a large interconnect layer such as a metal layer. There are generally no such performance issues with elements like diodes and vias, however.

See also [DRC Incremental Connect](#), [DRC Tolerance Factor NAR](#), [Net Area Ratio Accumulate](#), and [Net Area Ratio Print](#).

### Examples

#### Example 1

Basic layer derivations:

```
// Select metall1 polygons in nets also having layer pad:  
x = NET AREA RATIO metall1 pad > 0  
  
// Select metall1 polygons which are not in nets having layer diode:  
y = NET AREA RATIO metall1 diode == 0
```

#### Example 2

The following is a simple antenna check:

```
gate = poly AND diff  
CONNECT ipoly gate  
ANT.poly {  
@ Ratio of interconnect poly area to the gate area > 100  
NET AREA RATIO ipoly gate > 100  
}
```

#### Example 3

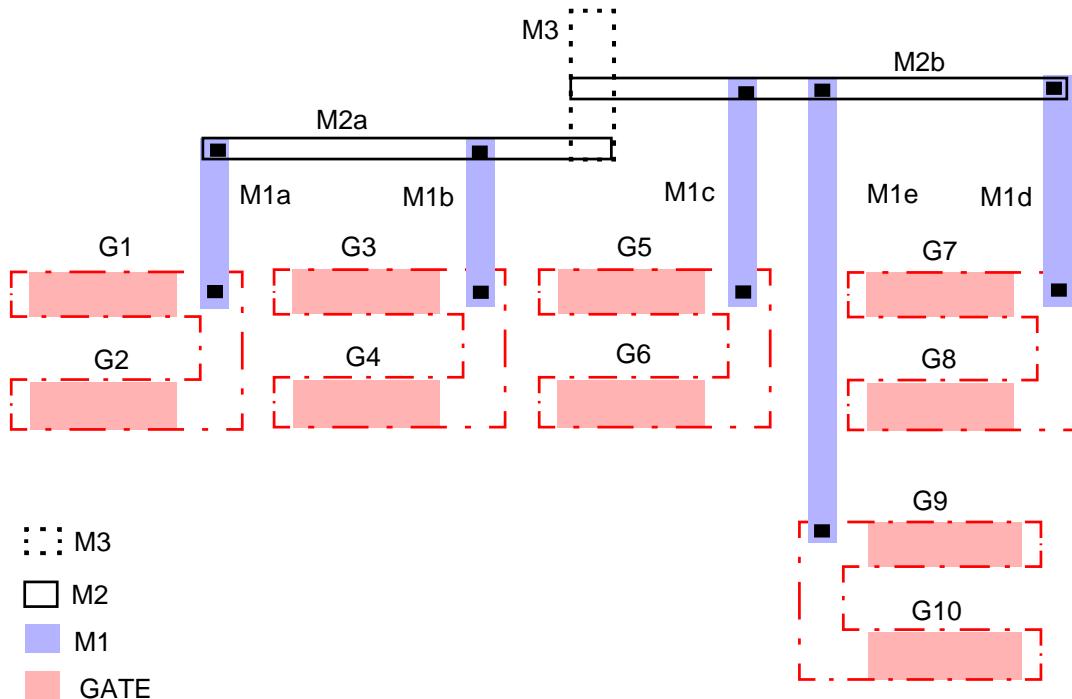
Using DRC Incremental Connect YES, you can do more sophisticated antenna checking:

```
DRC INCREMENTAL CONNECT YES  
gate = poly AND diff  
sd = diff NOT gate  
CONNECT poly gate  
ANT.poly {@ Ratio of poly area to the gate area > 100  
NET AREA RATIO poly gate > 100 }  
  
CONNECT met1 poly sd BY cont  
  
ANT.met1 {@ Ratio of metall1 side wall area (perimeter * 0.5  
@ thickness) to gate area > 50  
NET AREA RATIO met1 gate > 50 [PERIMETER(met1)*0.5/AREA(gate)] }
```

**Example 4**

The following example demonstrates Net Area Ratio accumulation. [Figure 4-175](#) represents nets of a layout. The M3, M2, and M1 objects correspond to metal paths. The G objects correspond to gates.

**Figure 4-175. Net Area Ratio Accumulation**



At the metal1 / poly level, assume that M1a, M1b, M1c, M1d, and M1e are all on different nodes from each other. The statements:

```
DRC INCREMENTAL CONNECT YES
gate = poly AND diff
CONNECT m1 poly BY cont
ac1 = NET AREA RATIO m1 gate >= 0 ACCUMULATE RDB ac1.rdb m1 gate BY LAYER
```

compute the antenna ratios between M1 and GATE and output these ratios with the gate shapes in the NARAC layer AC1 (notice the constraint:  $\geq 0$ ).

Assume, for future reference, that the ratios at this level are:

- Area M1a / Area (G1 and G2) = 10 is assigned to G1 and G2
- Area M1b / Area (G3 and G4) = 20 is assigned to G3 and G4
- Area M1c / Area (G5 and G6) = 30 is assigned to G5 and G6
- Area M1d / Area (G7 and G8) = 40 is assigned to G7 and G8
- Area M1e / Area (G9 and G10) = 50 is assigned to G9 and G10

## Net Area Ratio

---

At the metal2 level, assume that M2a and M2b are on different nodes. The statements:

```
CONNECT m2 m1 BY via1  
ac2 = NET AREA RATIO m2 gate >= 0 ACCUMULATE ac1 RDB ac2.rdb m2 gate  
BY LAYER
```

first compute the antenna ratios between M2 and GATE. These ratios are added to those in AC1 to create the new NARAC AC2. Assume the ratios at M2 are:

Area M2a / Area (G1, G2, G3 and G4) = 100 is assigned to G1 through G4  
Area M2b / Area (G5, G6, G7, G8, G9 and G10) = 200 is assigned to G5 through G10

The accumulated ratios in the NARAC AC2 are:

G1, G2 = 100 + 10, or 110  
G3, G4 = 100 + 20, or 120  
G5, G6 = 200 + 30, or 230  
G7, G8 = 200 + 40, or 240  
G9, G10 = 200 + 50, or 250

At the metal3 level, there is one node. Assuming the connect statements are done as shown, the next operation outputs all gates having accumulated antenna ratios greater than 300:

```
CONNECT m3 m2 by via2  
rule { NET AREA RATIO m3 gate > 300 ACCUMULATE ac2 }
```

If the antenna ratio at metal3 is 70 (all gates), then the accumulated antenna ratios from AC2 give:

G1, G2 = 110 + 70, or 180  
G3, G4 = 120 + 70, or 190  
G5, G6 = 230 + 70, or 300  
G7, G8 = 240 + 70, or 310  
G9, G10 = 250 + 70, or 320

and the output of the rule is gates G7, G8, G9, and G10.

### Example 5

The following example illustrates the use of the BY LAYER keyword in conjunction with the Results Database feature:

```
ANT.M1 = {  
@[ (MET1 perimeter * height)/(GATE area)] - DIODE count * 20  
@ > 75  
NET AREA RATIO met1 poly sd gate > 75  
[(PERIMETER(met1) * 0.5 / AREA(gate)) - COUNT(sd) * 20]  
RDB ant_m1_out.db met1 poly sd gate BY LAYER }
```

The RDB output from this command is a separate file, ant\_m1\_out.db. In this file, each antenna error is broken down by net; the polygon layers associated with the net are then presented on a layer-by-layer basis, for MET1, POLY, SD, and GATE. With each layer polygon is the associated value calculated for perimeter, area, or count, respectively.

## Net Area Ratio Accumulate

Layer operation

**NET AREA RATIO ACCUMULATE** *layer1 layer2 constraint* ‘[*expression*]’

### Summary

This operation selects all polygons from *layer1* that, after computation of the *expression*, satisfy the *constraint*. The computed value from the *expression* is attached to the output layer.

### Parameters

- *layer1*

A required original layer, layer set, or a derived polygon layer. This layer must be derived from a previous **Net Area Ratio** operation specified with the ACCUMULATE keyword or a previous Net Area Ratio Accumulate operation.

- *layer2*

A required original layer, layer set, or a derived polygon layer. This layer must be derived from a previous **Net Area Ratio** operation specified with the ACCUMULATE keyword or a previous Net Area Ratio Accumulate operation.

- ‘[*expression*]’

A required numeric expression within square brackets that allows customizable control over the operation computations. The expression defines calculations to be made using the input layers.

The *expression* may involve numbers (including numeric variables), operators, parentheses ( ), and the functions given in [Table 4-32](#):

**Table 4-32. Net Area Ratio Accumulate Math Functions**

Function	Definition
VALUE( <i>alayer</i> )	The accumulated ratio value associated with the <i>alayer</i> . The <i>alayer</i> must be an input layer to the operation.
SQRT(x)	square root of x
EXP(x)	exponential (base e) of x
LOG(x)	natural logarithm of x
SIN(x)	sine of x in radians
COS(x)	cosine of x in radians
TAN(x)	tangent of x in radians

**Table 4-32. Net Area Ratio Accumulate Math Functions**

Function	Definition
MIN( <i>expression1,expression2</i> )	Returns the lesser value of the two expressions. The expressions must be of the same form as the primary Net Area Ratio <i>expression</i> , without the square brackets.
MAX( <i>expression1,expression2</i> )	Returns the maximum value of the two expressions. The expressions must be of the same form as the primary Net Area Ratio <i>expression</i> , without the square brackets.

In Table 4-32, x is a numeric argument (including numeric variables and other numeric-valued expressions). There is no compile-time or runtime exception checking on the arguments of these functions, and unreasonable or undefined values may result from certain arguments. The *alayer* is an input layer in the same Net Area Ratio Accumulate operation and is an accumulation layer, otherwise a compilation error results.

The expression must not result in strictly negative values because such values cannot be checked by a *constraint*. Division by zero does *not* satisfy any *constraint*.

Parentheses have the highest precedence for the calculation of numeric values and may be used for grouping of terms.

**Operators** — Unary operators (+, -, !, ~) require only one numerical argument and all such operators have the same precedence. The + and - operators are the usual positive and negative signs. The other unary operators do the following:

**! operator** — Returns 0 (or false) if its argument is non-zero and 1 (or true) if its argument is 0.

**~ operator** — Returns 0 (or false) if the argument is positive and 1 (or true) if the argument is non-positive.

Table 4-3 shows some useful combinations of the ! and ~ operators.

The binary operators ^, \*, /, +, - require two numerical arguments. The ^ operator is the same as the C language pow( ) function, where x ^ y means x raised to the y power. The \*, /, +, and - operators are multiplication, division, addition, and subtraction, respectively, and have the customary precedence. The ^ operator has the same precedence as \* and /.

The binary operators || (OR) and && (AND) are the same as the C language operators of the same type, except that they have the same precedence as binary + and -. They expect numeric-valued inputs. For example:

VALUE(ac1) && VALUE(ac2) returns 1 if both inputs are non-zero and 0 otherwise.

VALUE(ac1) || VALUE(ac2) returns 1 if either input is non-zero and 0 otherwise.

- ***constraint***

A required string that is one of the constraints listed in the “Constraint Notation” or “Alternate Notation” column of [Table 2-2](#) on page 45. It is interpreted as the constraining value of the operation.

The tolerance for comparing the ***constraint*** to the value computed by the operation is controlled by the [DRC Tolerance Factor NAR](#) statement. The default tolerance is 0.005.

## Description

This operation selects all polygons from ***layer1*** that, after computation of the ***expression***, satisfy the ***constraint***. The computed value from the ***expression*** is attached to the output layer.

This operation is useful when you want to find calculated values from previously-derived Net Area Ratio accumulation (NARAC) layers. Parameters ***layer1*** and ***layer2*** must each be NARAC layers; that is, they must each be the output layer of a previous Net Area Ratio operation with ACCUMULATE specified, or a Net Area Ratio Accumulate operation. In addition, they must either have the same layer of origin or be geometrically equivalent. The operation creates a NARAC layer consisting of a subset of the polygons in ***layer1*** with potentially new values attached.

The ***expression*** is similar to the *expression* in the [Standard Form](#) of the Net Area Ratio operation, except that the only function that may be computed for the input layers is VALUE. This function returns the value attached to the polygon(s) for which the ***expression*** is computed.

The Net Area Ratio Accumulate operation proceeds as follows:

1. For each polygon  $P_1$  in ***layer1***, let  $V_1$  be its attached value that it received from a Net Area Ratio accumulation operation. Similarly, let  $P_2$  be the polygon in ***layer2***, *coincident with  $P_1$* , and let  $V_2$  be its attached value.
2. Define  $V_2$  to be zero if  $P_2$  does not exist.
3. Evaluate the ***expression*** using  $V_1$  and  $V_2$ .
4. If the ***expression*** does not result in division by zero, and the value  $V$  of the ***expression*** satisfies the ***constraint***, then  $P_1$  is output with  $V$  attached.

Be aware of the condition in step 1 that polygons  $P_2$  and  $P_1$  are coincident.

As an example of the combined use of the Net Area Ratio operation using ACCUMULATE and a Net Area Ratio Accumulate operation:

```
// For each individual gate, the ratio of the area of all
// metall on the same net as the gate to the individual
// area of the gate cannot exceed 100.

DRC INCREMENTAL CONNECT YES
CONNECT gate
X = NET AREA RATIO gate >= 0 [ AREA( gate ) ] ACCUMULATE
CONNECT poly gate
CONNECT metall poly BY contact
Y = NET AREA RATIO metall gate >= 0 [ AREA( metall ) ]
ACCUMULATE
```

```
rule { NET AREA RATIO ACCUMULATE X Y > 100
    [ VALUE( Y ) / VALUE( X ) ]
// The values previously assigned to X and Y polygons by
// Net Area Ratio are used here. The resulting computed values
// from this rule are attached to polygons on layer X.
}
```

For hierarchical applications, all NARAC layers are treated hierarchically.

The following are notes on two of the operators:

**! operator** — Returns “0” (or false) if its argument is non-zero and “1” (or true) if its argument is zero. You can use successive ! operators, for example: !!VALUE(ac1). If VALUE(ac1) is not 0, !VALUE(ac1) returns “0”. Therefore, !!VALUE(ac1) would return a “1”. In effect !!VALUE(ac1) acts as a detector of whether or not ac1 polygons have any accumulated value.

**~ operator** — Returns “0” (or false) if the argument is positive and “1” (or true) if the argument is non-positive. Recall, the ! operator returns “0” if the argument is positive or negative and returns “1” if the argument is zero. The ~ operator is particularly useful where the sign of the argument is of concern.

Table 4-3 shows some useful combinations of the ! and ~ operators.

See also [DRC Incremental Connect](#), [DRC Tolerance Factor NAR](#), and [Net Area Ratio Print](#).

## Examples

### Example 1

Here as a basic example that determines the area ratio of poly to gate and the area ratio of metal1 to gate. If the sum of those area ratios exceeds the constraint, then the gates are output.

```
DRC INCREMENTAL CONNECT YES

CONNECT poly gate
// find ratio of poly to gate areas and attach to gates
X = NET AREA RATIO poly gate >= 0
    [(AREA(poly)-AREA(gate))/AREA(gate)] ACCUMULATE

CONNECT metal1 poly BY contact
// find ratio of metal1 to gate areas and attach to gates
Y = NET AREA RATIO metal1 gate >= 0
    [AREA(metal1)/AREA(gate)] ACCUMULATE

// if the combined accumulation values on X and Y exceed 20, output X
// with the new value.
Z = NET AREA RATIO ACCUMULATE X Y > 20 [VALUE(X)+VALUE(Y)]
...
```

# Net Area Ratio Print

Layer operation

**NET AREA RATIO PRINT** *layer filename*

## Parameters

- *layer*

A required derived polygon layer that must be the output of a [Net Area Ratio](#) operation using ACCUMULATE or a [Net Area Ratio Accumulate](#) operation.

- *filename*

A required string that specifies a filename for the output ASCII file. The *layer* must be specified before the *filename* to prevent ambiguity. Subdirectories in a pathname are created as required.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

## Description

Prints the input net area ratio accumulation (NARAC) information to the specified file.

Each line in the destination file corresponds to a polygon in the input *layer*, and consists of its lower left vertex coordinates followed by the antenna ratio associated with the polygon. These are specified in user units. Here is an example of the *filename* output:

```
189.625 406 23.7767
217.625 68 20.814
```

This operation must be executed for printing to occur. For example, it can be an output-generating operation in a rule check:

```
rule { x = NET AREA RATIO m3 gate3 > 300 ACCUMULATE AC2
      NET AREA RATIO PRINT x '/usr/illmap.nar.output'
}
```

Output from Net Area Ratio Print operations is different from [Net Area Ratio](#) RDB output in the following ways:

- Net Area Ratio Print does not generate a nmDRC results style database like Net Area Ratio RDB output.
- Net Area Ratio Print uses only NARAC layers for input and for output statistics. Net Area Ratio RDB allows you to specify non-ACCUMULATE layers for output.

## Examples

In this example, *layer1* would be derived by a previous Net Area Ratio ACCUMULATE operation. The rule is used to output the accumulation data to the specified file.

```
rule {NET AREA RATIO PRINT layer1 "/usr/illmap.nar.output"}
```

## Net Interact

Layer operation

**NET INTERACT** *layer1 layer2 constraint*

### Parameters

- ***layer1***

A required original layer or layer set, or a derived polygon layer. Connectivity must be established on this layer.

- ***layer2***

A required original layer or layer set, or a derived polygon layer. Connectivity must be established on this layer.

- ***constraint***

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

The ***constraint*** is interpreted as the number of nodes a polygon from ***layer1*** must interact with on ***layer2*** in order to be output.

### Description

Selects polygons by net from ***layer1*** that interact with polygons on ***layer2***, where the total number of nodes that a ***layer1*** net interacts with meets the ***constraint***. Connectivity information must exist on both input layers. The [Interact](#) operation defines polygon interaction.

Here is a precise definition. Consider this operation:

```
Z = NET INTERACT X Y constraint
```

For each net N on layer X:

1. Let  $X_N$  be the set of all polygons from X on net N.
2. Let  $Y_N = Y \text{ INTERACT } X_N$ . Connectivity is passed from Y to  $Y_N$ .
3. Define C to be the number of distinct nodes in  $Y_N$ . If C satisfies the ***constraint***, then add  $X_N$  to Z.

That is, when the total number of nodes from ***layer2*** polygons that interact with ***layer1*** polygons on a given (different from ***layer2***) node meets the ***constraint***, then the ***layer1*** polygons are output.

The output layer is empty if either of the input layers are empty. This operation is a node-preserving layer constructor.

See also [Net](#), [Not Net](#), and [Not Interact](#).

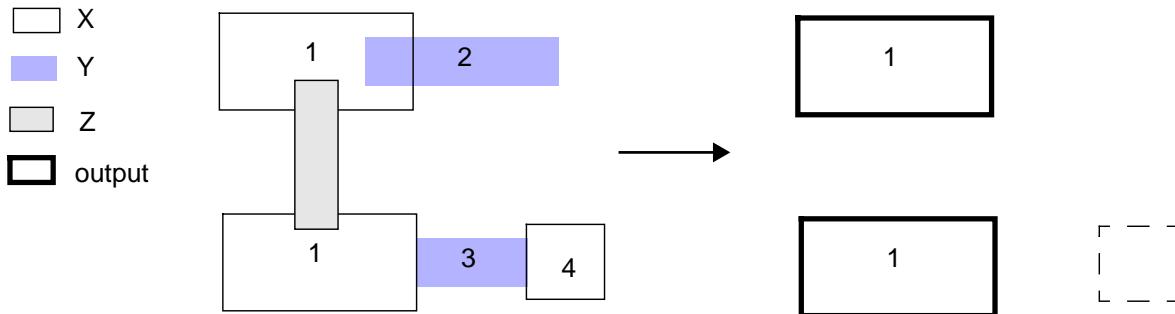
## Examples

### Example 1

Connectivity is required on the input layers. Polygons on a given node from X are output (with node IDs) based upon the number of different nodes on Y that interact with polygons on X.

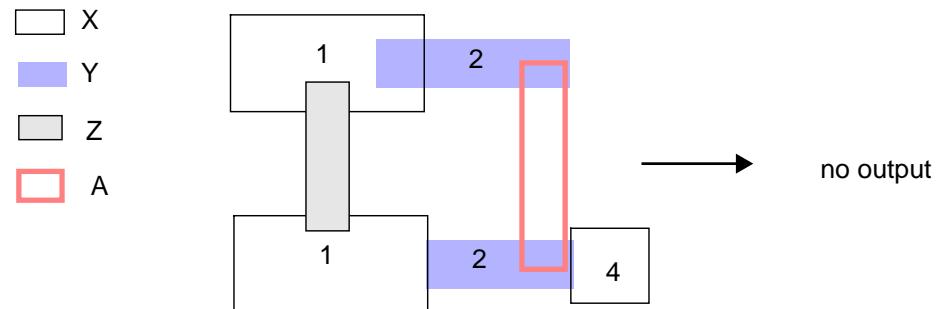
```
CONNECT X Z
CONNECT Y
rule { NET INTERACT X Y > 1 }
```

**Figure 4-176. Net Interact**



### Example 2

```
CONNECT X Z
CONNECT Y A
rule { NET INTERACT X Y > 1 }
```



## NOT

Layer operation

**NOT** *layer1 layer2*

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.

### Description

Selects all *layer1* polygon areas not common to *layer2* polygons. If *layer2* is empty, the NOT operation generates output equivalent to the polygon data on *layer1*.

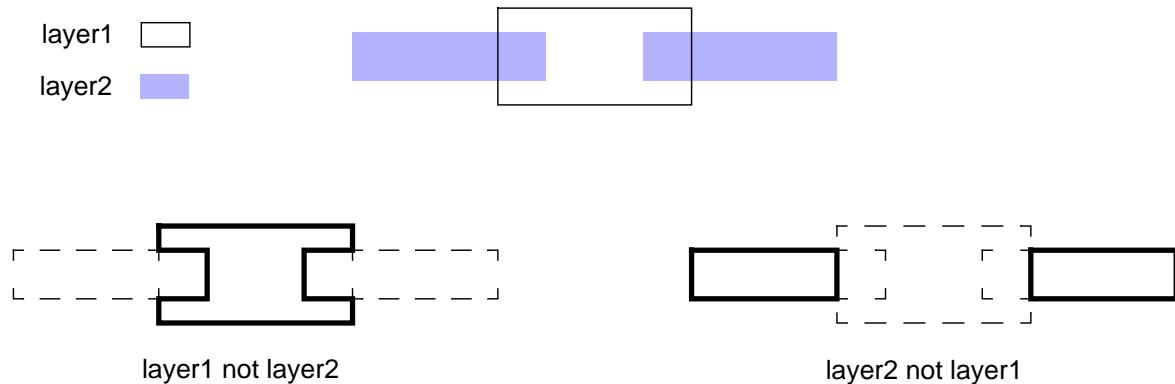
The NOT operation is node-preserving for connectivity extraction. Connectivity is generally passed from *layer1* to the output layer. See “[Node-Preserving Layer Operations](#)” in the *Calibre Verification User’s Manual* for details.

See also [AND](#), [OR](#), and [XOR](#).

### Examples

Figure 4-177 shows two Boolean NOT operations. The left operation selects all *layer1* polygon area not common to *layer2* polygons. The right operation selects all *layer2* polygon area not common to *layer1* polygons.

**Figure 4-177. Boolean NOT**



## Not Angle

Layer operation

### NOT ANGLE *layer constraint*

#### Parameters

- ***layer***

A required original layer or layer set, or a derived polygon or edge layer.

- ***constraint***

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

The constraint must contain floating-point numbers and is interpreted in degrees. It must be greater than or equal to 0 but less than or equal to 90. You cannot specify the constraint  $> 90$ .

#### Description

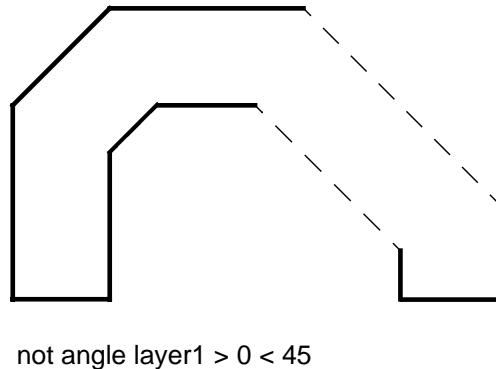
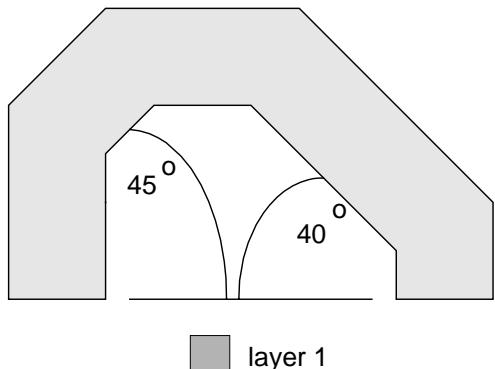
Selects all ***layer*** edges having a smaller angle magnitude with respect to the x-axis of the coordinate system, which do not conform to the ***constraint***. This is the complement of the [Angle](#) operation.

Hierarchical Calibre applications perform this operation hierarchically in all cases.

#### Examples

##### Example 1

This operation selects all layer1 edges that do not conform to the constraint.



##### Example 2

This example selects only edges orthogonal to the database axes.

```
orthogonal_metal_edges = NOT ANGLE metal > 0 < 90
```

### Example 3

This example selects gates that have angled edges.

```
non-aligned-gate {  
    @ Gates must be horizontal along the edge inside polysilicon  
    @ and vertical along the edge inside diffusion. Flag gates  
    @ themselves, not gate edges.  
    // derive gate  
    gate = AND poly diff  
    // derive gate edges in poly and diffusion  
    gate_edge_in_poly = COINCIDENT EDGE gate diff  
    gate_edge_in_diff = NOT COINCIDENT EDGE gate diff  
    // find non-horizontal gate edges in poly  
    bad_angle_1 = NOT ANGLE gate_edge_in_poly == 0  
    // find non-vertical gate edges in diff  
    bad_angle_2 = NOT ANGLE gate_edge_in_diff == 90  
    // flag the bad gates, not the edges  
    bad_gate_1 = gate WITH EDGE bad_angle_1  
    bad_gate_2 = gate WITH EDGE bad_angle_2  
    OR bad_gate_1 bad_gate_2 // Remove duplicate errors.  
}
```

## Not Area

Layer operation

### NOT AREA *layer constraint*

#### Parameters

- ***layer***  
A required original layer or layer set, or a derived polygon layer.
- ***constraint***  
A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.  
The constraint is interpreted in user units squared.

#### Description

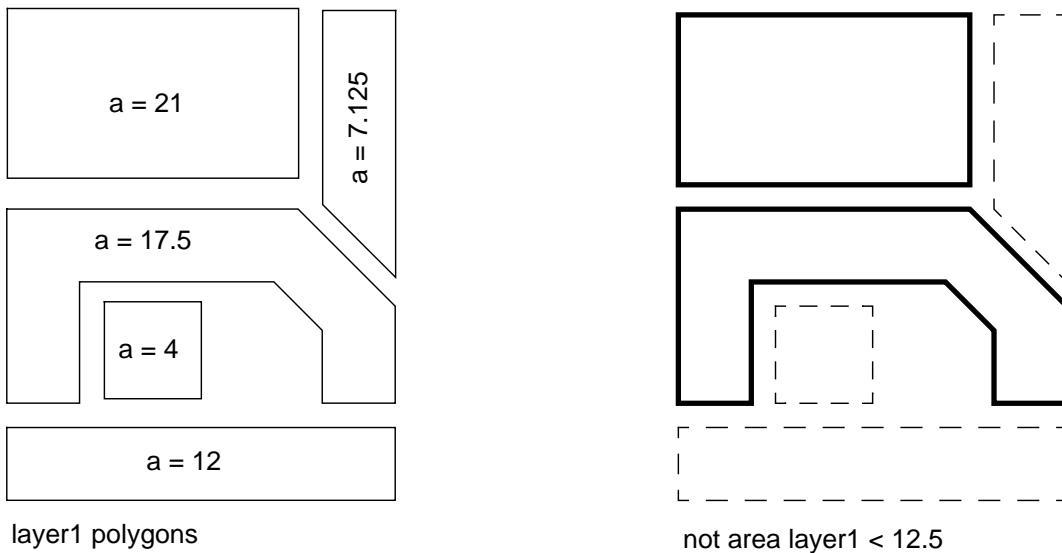
Selects all ***layer*** polygons having areas that do not conform to the given ***constraint***. This is the complement of the [Area](#) operation. Depending on your methodology, a narrow constraint range like  $> a < b$ , may be preferable to a constraint like  $== a$ .

#### Examples

##### Example 1

The Not Area operation shown in Figure 4-178 selects all layer1 polygons not conforming to the constraint.

**Figure 4-178. Not Area**



##### Example 2

```
// Check that area of metal is greater than 3.2 square microns but not
// greater than 5.6 square microns.
metal_area_check { NOT AREA metal > 3.2 <= 5.6 }
```

## Not Coincident Edge

Layer operation

**NOT COINCIDENT EDGE *layer1 layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.

### Description

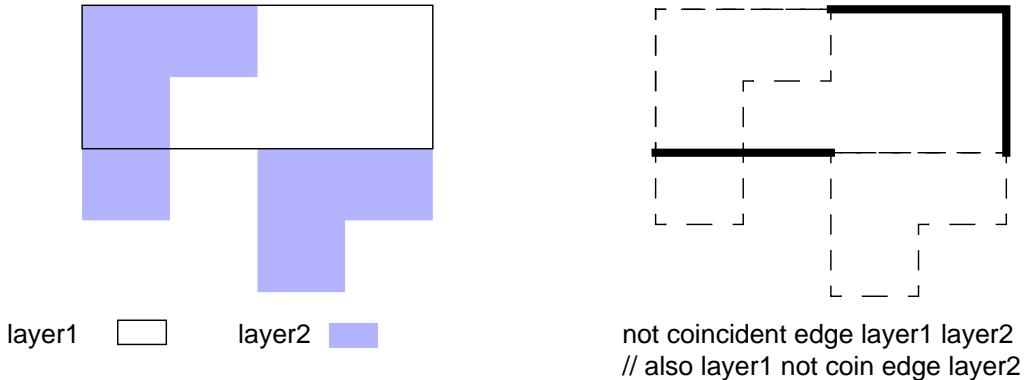
Selects all *layer1* edges or edge portions that are not coincident with any *layer2* edges. This is the complement of the [Coincident Edge](#) operation. If *layer2* is empty, the operation generates output equivalent to the edge data on *layer1*.

See also [Not Coincident Inside Edge](#), [Not Coincident Outside Edge](#), [Not Inside Edge](#), [Not Outside Edge](#), and [Not Touch Edge](#).

### Examples

The Not Coincident Edge operation shown in Figure 4-179 selects all *layer1* edges or edge segments that are not coincident with *layer2* edges.

**Figure 4-179. Not Coincident Edge**



## Not Coincident Inside Edge

Layer operation

**NOT COINcident INside EDGE *layer1 layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.

### Description

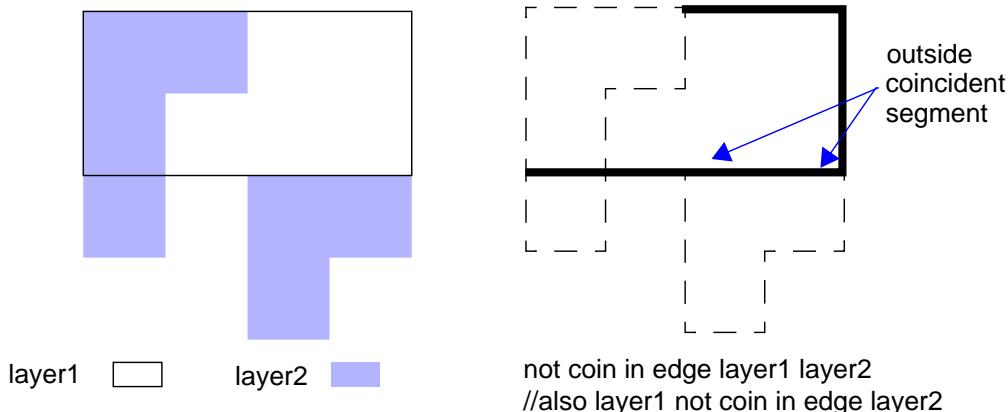
Selects all *layer1* edges or edge portions that are not coincident with any *layer2* edges, but including any coincident edge segments where there is no adjacent shared interior polygon area. This operation is the complement of the [Coincident Inside Edge](#) operation. If *layer2* is empty, the operation generates output equivalent to the edge data on *layer1*.

See also [Not Coincident Edge](#), [Not Coincident Outside Edge](#), [Not Touch Inside Edge](#), and [Not Inside Edge](#).

### Examples

The Not Coincident Inside Edge operation shown in Figure 4-180 selects all *layer1* edges or edge segments that are not inside-coincident with *layer2* edges.

**Figure 4-180. Not Coincident Inside Edge**



## Not Coincident Outside Edge

Layer operation

**NOT COINcident OUTside EDGE *layer1* *layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.

### Description

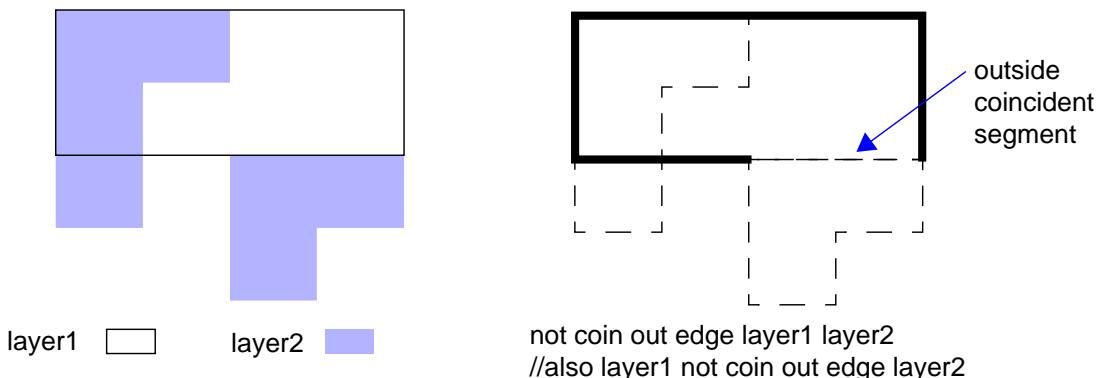
Selects all *layer1* edges or edge portions that are not coincident with any *layer2* edges, but including coincident edge segments where there is shared internal polygon area. This operation is the complement of the [Coincident Outside Edge](#) operation. If *layer2* is empty, the operation generates output equivalent to the edge data on *layer1*.

See also [Not Coincident Edge](#), [Not Coincident Inside Edge](#), [Not Outside Edge](#), and [Not Touch Outside Edge](#).

### Examples

The Not Coincident Outside Edge operation shown in Figure 4-181 selects all *layer1* edges or edge segments that are not outside-coincident with any *layer2* edges.

**Figure 4-181. Not Coincident Outside Edge**



## Not Cut

Layer operation

**NOT CUT** *layer1* *layer2* [*constraint* [BY NET] [EVEN | ODD]]

### Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- ***layer2***  
A required original layer or layer set, or a derived polygon layer.
- ***constraint***  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. This constraint specifies the number of *layer2* polygons or nets that a *layer1* polygon cannot share any of its area with, in order for the *layer1* polygon to be selected. The constraint must contain non-negative integers.
- **BY NET**  
An optional keyword used with the *constraint* parameter that specifies a *layer1* polygon is selected when the number of distinct nets in the set of *layer2* polygons, which share some of their area with the *layer1* polygon, does not meet the specified constraint. The connectivity of *layer2* is required and is checked at compilation time.
- **EVEN**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an even number. May not be used with ODD.
- **ODD**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an odd number. May not be used with EVEN.

### Description

Selects all *layer1* polygons that share all or none of their internal areas with *layer2* polygons. This is the complement of the [Cut](#) operation. If a *constraint* is present, the *layer1* polygons that do not meet the constraint are output. If *layer2* is empty, the Not Cut operation generates output equivalent to the polygon data on *layer1*.

### BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that do not meet the constraint are output.

See also [Not Interact](#).

## Not Cut

---

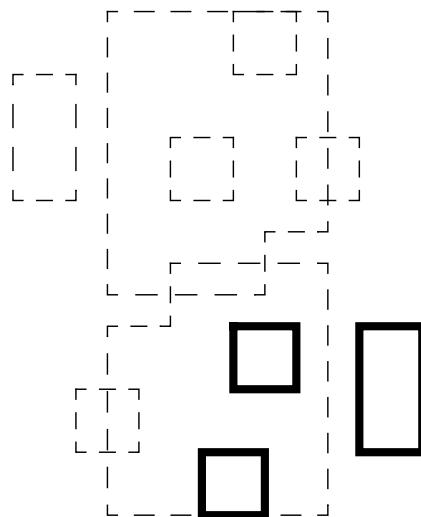
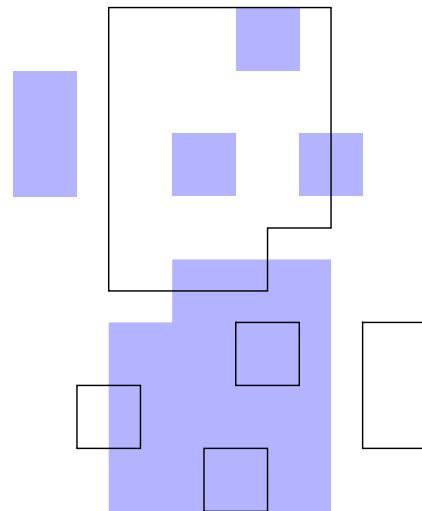
### Examples

Figure 4-182 shows two Not Cut operations. The operation on the left selects only layer1 polygons that share all or none of their area with layer2 polygons. The operation on the right selects only layer1 polygons that share all, some, or none of their area with layer2 polygons.

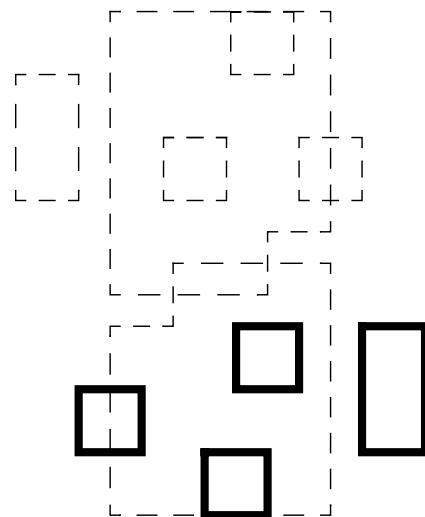
**Figure 4-182. Not Cut**

layer1

layer2



not cut layer1 layer2  
//also layer1 not cut layer2



not cut layer1 layer2 > 1

## Not Donut

Layer operation

**NOT DONUT** *layer* [*constraint*]

### Parameters

- *layer*  
A required original layer, layer set, or a derived polygon layer.
- *constraint*  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. Specifies the number of interior cycles a *layer* polygon must not have in order to be selected.

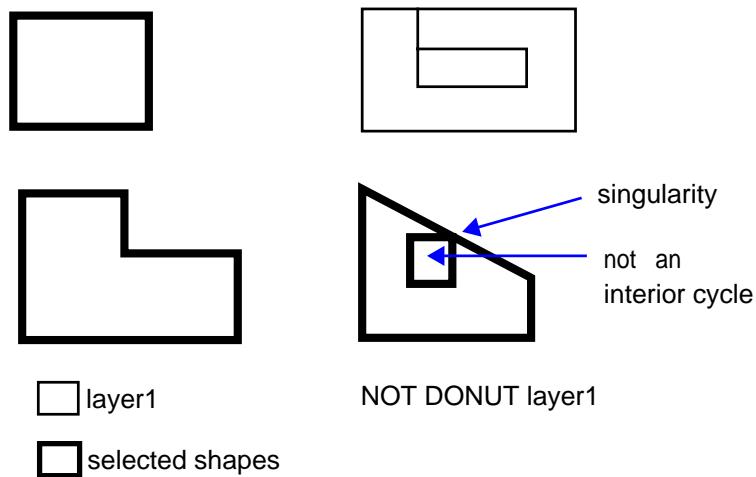
### Description

Selects all *layer* polygons that do not have interior cycles (or holes). If a *constraint* is used, polygons that do not meet the constraint are selected. This is the complement of the [Donut](#) operation (which see for a complete description of cycles).

### Examples

Figure 4-183 shows where all layer1 polygons that do not have interior cycles are selected by the rule check.

**Figure 4-183. Not Donut**



## Not Enclose

Layer operation

**NOT ENCLOSE** *layer1 layer2* [*constraint* [BY NET] [EVEN | ODD]]

### Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- ***layer2***  
A required original layer or layer set, or a derived polygon layer.
- ***constraint***  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. This constraint specifies the number of *layer2* polygons a *layer1* polygon does not completely enclose in order to be selected. The constraint must contain non-negative integers.
- **BY NET**  
An optional keyword, used with the *constraint* parameter, that specifies a *layer1* polygon is selected when the number of distinct nets on *layer2*, enclosed by the *layer1* polygon, does not meet the specified *constraint*. The connectivity of *layer2* is required and is checked at compilation time.
- **EVEN**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an even number. May not be used with ODD.
- **ODD**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an odd number. May not be used with EVEN.

### Description

Selects all *layer1* polygons that contain no *layer2* polygons as a subset (that is, a logical OR of a *layer1* and *layer2* polygon(s) does not yield exactly the *layer1* polygon). This is the complement of the [Enclose](#) operation. Can also select all *layer1* polygons that do not completely enclose a specified number of *layer2* polygons, as indicated by a *constraint*. If *layer2* is empty, the Not Enclose operation generates output equivalent to the polygon data on *layer1*.

### BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that do not meet the *constraint* are output.

See also [Not Enclose Rectangle](#), [Not Inside](#), and [Not Interact](#).

## Examples

Figure 4-184 shows two Not Enclose operations. The operation on the left selects all layer1 polygons that do not completely enclose a layer2 polygon. The operation on the right selects only the layer1 polygons that do not completely enclose more than one layer2 polygon.

**Figure 4-184. Not Enclose**



## Not Enclose Rectangle

Layer operation

**NOT ENCLOSURE RECTANGLE *layer width length* [ORTHOGONAL ONLY]**

### Parameters

- ***layer***  
A required original or derived polygon layer.
- ***width***  
A required positive floating-point number interpreted in user units.
- ***length***  
A required positive floating-point number interpreted in user units.
- **ORTHOGONAL ONLY**  
An optional keyword which limits selection to polygons that do not enclose rectangles of the given ***width*** and ***length***, and having sides that are parallel (respectively) to the coordinate axes of the database.

### Description

Selects all polygons on the specified ***layer*** that cannot enclose a rectangle of the specified ***width*** and ***length***. This is the complement of the [Enclose Rectangle](#) operation. Edge coincidence between the enclosing polygon and an enclosed rectangle does not satisfy the Not Enclose Rectangle condition.

The tool may align ***width*** along either the x- or y-axis, but considers only enclosed rectangles with edges that are not parallel to the database axes or are oriented at 45 degrees with respect to the database axes. To consider only enclosed rectangles having sides that are parallel to the coordinate axes, you must specify ORTHOGONAL ONLY.

The operation behaves exactly the same if ***width*** and ***length*** are interchanged. For example, the statement:

```
NOT ENCLOSURE RECTANGLE poly 8 2
```

produces the same results as the statement:

```
NOT ENCLOSURE RECTANGLE poly 2 8
```

See also [Not Enclose](#).

### Examples

```
// show all polygons on metal that cannot enclose a
// 2 x 4 rectangle that is orthogonal to the database axes or
// is oriented at 45 degrees to the database axes.
rule { NOT ENCLOSURE RECTANGLE metal 2 4 }
```

## Not Inside

Layer operation

**NOT INside *layer1* *layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.

### Description

Selects all *layer1* polygons that do not share all of their areas with *layer2* polygons. This is the complement of the [Inside](#) operation. If a *layer1* polygon shares all of its area with a *layer2* polygon and the two polygons have coincident edges, the Not Inside condition is not satisfied. If *layer2* is empty, the Not Inside operation generates output equivalent to the polygon data on *layer1*.

See also [Not Enclose](#), [Not Inside Cell](#), [Not Inside Edge](#), [Not Interact](#), and [Outside](#).

## Not Inside

---

### Examples

Figure 4-185 shows two Not Inside operations. The operation on the left selects all layer1 polygons that do not share all of their areas with layer2 polygons. The operation on the right reverses the layer order, therefore, it selects all layer2 polygons that do not share all of their areas with layer1 polygons.

**Figure 4-185. Not Inside**



## Not Inside Cell

Layer operation

**NOT INSIDE CELL** *layer* *cell\_name* [*cell\_name* ...] [PRIMARY ONLY]  
 [WITH MATCH [GOLDEN] [RULE *rule\_name*] ...]  
 [WITH LAYER *layer2*]

Used primarily in hierarchical Calibre applications.

### Summary

Selects all polygons from *layer* that are not inside the cells you specify, or the subhierarchies of the specified cells.

### Parameters

- ***layer***  
 A required original layer.
- ***cell\_name***  
 A required name of a cell. You can specify *cell\_name* any number of times in one statement. Cell names are case-sensitive. The *cell\_name* can be a string variable (see [Variable](#)).  
 The *cell\_name* parameters can contain one or more asterisk (\*) wildcard characters, where the \* character matches zero or more characters. When using \*, enclose the cell name in quotes; otherwise, a compilation error occurs because the asterisk is a reserved symbol.  
 The string CALIBRE\_TOP\_CELL (case-insensitive) is understood to be the top-level cell in the database.
- PRIMARY ONLY  
 An optional keyword that instructs the tool to include only shapes that exist outside the top levels of cells appearing as *cell\_name* parameters.
- WITH MATCH [GOLDEN] [RULE *rule\_name*] ...  
 Optional keyword set that allows a placed cell to be treated as a *cell\_name* parameter to the operation if this cell geometrically matches another cell (unplaced) that is already specified as a *cell\_name* parameter. The WITH MATCH keyword is particularly useful for matching cells to so-called “golden” cells, when the exact name correspondence may not be known. This keyword set may only be used in hierarchical Calibre applications. The following additional keywords may be specified:
  - GOLDEN  
 Causes the operation to attempt to match only cells that appear in designs specified with [Layout Path](#) or [Layout Path2](#) statements that also use the GOLDEN keyword. The GOLDEN keyword may only be specified with the WITH MATCH keyword.

- RULE *rule\_name*

Causes the operation to use [Layout Cell Match Rule](#) statements having a matching *rule\_name* in order to define the cells that are geometrically equal. The Layout Cell Match Rule algorithm is used instead of the default algorithm for determining geometrically equal cells. More than one RULE keyword set may be specified, and this keyword set may only be specified with WITH MATCH. Each *rule\_name* that matches a Layout Cell Match Rule statement is used. Duplications of *rule\_name* are ignored; failure to match a corresponding *rule\_name* causes an LCMR4 compiler error.

- WITH LAYER *layer2*

An optional keyword that limits selection to the specified cells that have any geometry on *layer2* (which must be an original layer) in their immediate hierarchy.

## Description

Selects all polygons from *layer* that are not inside the cells you specify, or the subhierarchies of the specified cells. This operation is the complement of [Inside Cell](#). The parameter order given must be observed to avoid ambiguity.

The tool selects only the shapes that are not instantiated inside of a cell. A shape that is inside the boundary of every placement of a cell, but at a higher level of hierarchy, is selected.

This statement is used primarily for hierarchical Calibre applications. It can be used in flat Calibre, but its usefulness is limited.

By default, a warning is issued if a *cell\_name* parameter cannot be found in the layout, but the run proceeds. You can control the severity of this case through the [Layout Input Exception Severity](#) INSIDE\_CELL setting.

The WITH MATCH keyword allows placed cells to be included as part of the Not Inside Cell calculation based upon geometric matching of another cell used as a *cell\_name* parameter. This feature should only be used with validated cells serving as *cell\_name* parameters in a hierarchical run. Note that specifying “non-golden” cells (cells that have not been previously validated) can result in warnings or errors during the Calibre run.

Before discussing how WITH MATCH works, the term *geometrically equal* is defined. Two cells A and B are said to be *geometrically equal* under the following conditions:

1. Flatten and merge *layer* in A and its sub-hierarchy into the coordinate space of A.
2. Flatten and merge *layer* in B and its sub-hierarchy into the coordinate space of B.
3. Perform a Boolean XOR of the results of Steps 1 and 2. If there are no results of the XOR, then A and B are geometrically equal.

This definition does not include text objects.

The output of WITH MATCH is as follows.

**NOT INSIDE CELL** *A<sub>1</sub> A<sub>2</sub> ... A<sub>m</sub>* **WITH MATCH** ...

where  $m \geq 1$ , is made equivalent to

**NOT INSIDE CELL layer  $B_1 B_2 \dots B_n$**

where  $n \geq 0$ . Each  $B_j$  is the name of a placed cell (the name may or may not be known) that is geometrically equal to some unplaced cell A whose name matches some  $A_i$  with (possibly) wildcards. The geometry on *layer* that does not appear in all such cells  $B_j$  is output.

The RULE *rule\_name* keyword causes the matching [Layout Cell Match Rule](#) *rule\_name* statement to be used to determine which cells are geometrically equal.

If the GOLDEN keyword is specified, then the unplaced cells  $A_i$  that the operation attempts to match come from designs specified in Layout Path[2] GOLDEN statements.

In ICVerify applications, this operation copies the input layer to the output layer and issues a warning.

## Examples

### Example 1

```
// select all metal polygons not inside ramcell and mem_blk
x = NOT INSIDE CELL metal ramcell mem_blk
```

### Example 2

This shows how the GOLDEN option might be used when golden cells are not to be concatenated with the primary design:

```
// Read in two layouts. Cell names are known for golden.oas, but not
// necessarily for layout.oas.
LAYOUT PATH layout.oas
LAYOUT PATH golden.oas GOLDEN // do not concatenate analog_block with
// layout.oas
...
// analog_block is a known cell name in golden.oas.
// Return all poly that is not in any cell that
// geometrically matches analog_block.
poly_outside_analog = NOT INSIDE CELL poly analog_block WITH MATCH GOLDEN
```

## Not Inside Edge

Layer operation

**NOT INside EDGE *layer1* *layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.

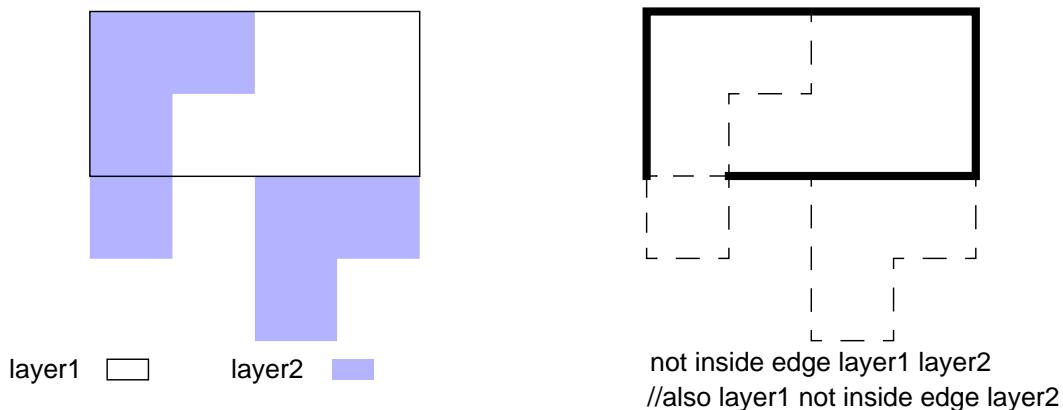
### Description

Selects all *layer1* edges or edge segments that do not lie completely inside *layer2* polygons. Also selects *layer1* edges or edge segments that are coincident with *layer2* edges. This is the complement of the [Inside Edge](#) operation. If *layer2* is empty, the Not Inside Edge operation generates output equivalent to the edge data on *layer1*. See also [Not Coincident Inside Edge](#), [Not Inside](#), [Outside Edge](#), and [Not Touch Inside Edge](#).

### Examples

The Not Inside Edge operation shown in Figure 4-186 selects all *layer1* edges that do not lie completely inside *layer2* polygons. Because coincident edges are not inside edges, they are selected also.

**Figure 4-186. Not Inside Edge**



## Not Interact

Layer operation

**NOT INTERACT** *layer1* *layer2* [*constraint* [BY NET] [EVEN | ODD]]  
[SINGULAR {ALSO | ONLY}]

### Summary

Selects polygons based upon the type and number of interactions they do not have with polygons on another layer.

### Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- ***layer2***  
A required original layer or layer set, or a derived polygon layer.
- ***constraint***  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint should contain non-negative integers. You specify the constraint to control the selection of *layer1* polygons according to the number of *layer2* polygons with which the interaction occurs.
- **BY NET**  
An optional keyword, used with the *constraint* parameter, which specifies the selection of a *layer1* polygon based upon the number of distinct nets (not polygons) it does not interact with. This occurs when the number of distinct nets in the set of *layer2* polygons interacting with a *layer1* polygon does not meet the specified *constraint*. The connectivity of *layer2* is required and is checked at compilation time.
- **EVEN**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an even number. May not be used with ODD.
- **ODD**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an odd number. May not be used with EVEN.
- **SINGULAR {ALSO | ONLY}**  
Optional keyword set that indicates how to handle point-to-point and point-to-edge interactions. If neither of these keyword sets are used, point-to-point and point-to-edge interactions are considered to satisfy the Not Interact condition. May not be specified with BY NET.

SINGULAR ALSO — Outside, point-to-point and point-to-edge interactions between *layer1* and *layer2* polygons are not considered for *layer1* output. If a *constraint* is specified, then *layer1* must not intersect the specified number of *layer2* polygons in order to be output.

SINGULAR ONLY — Selects all polygons from *layer1* that do not intersect polygons on *layer2* by outside point-to-point and point-to-edge interaction. If a *constraint* is specified, then *layer1* must not intersect the specified number of *layer2* polygons in point-to-point and point-to-edge interaction.

## Description

Selects all *layer1* polygons that do not share any area, edges, or edge segments with a *layer2* polygon. This is the complement of the [Interact](#) operation. If *layer2* is empty, the Not Interact operation generates output equivalent to the polygon data on *layer1*.

## BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that do not meet the *constraint* are output.

## Singular Keyword

The SINGULAR ALSO and SINGULAR ONLY keywords exclude point-to-point and point-to-edge interactions from being considered. In the either case, point-to-point and point-to-edge interactions may not be present for *layer1* polygon output to occur; however, in the latter case, other kinds of interactions cause *layer1* polygon output to occur.

See also [Not Cut](#), [Not Enclose](#), [Not Inside](#), [Not Touch](#), [Outside](#), and [Net Interact](#).

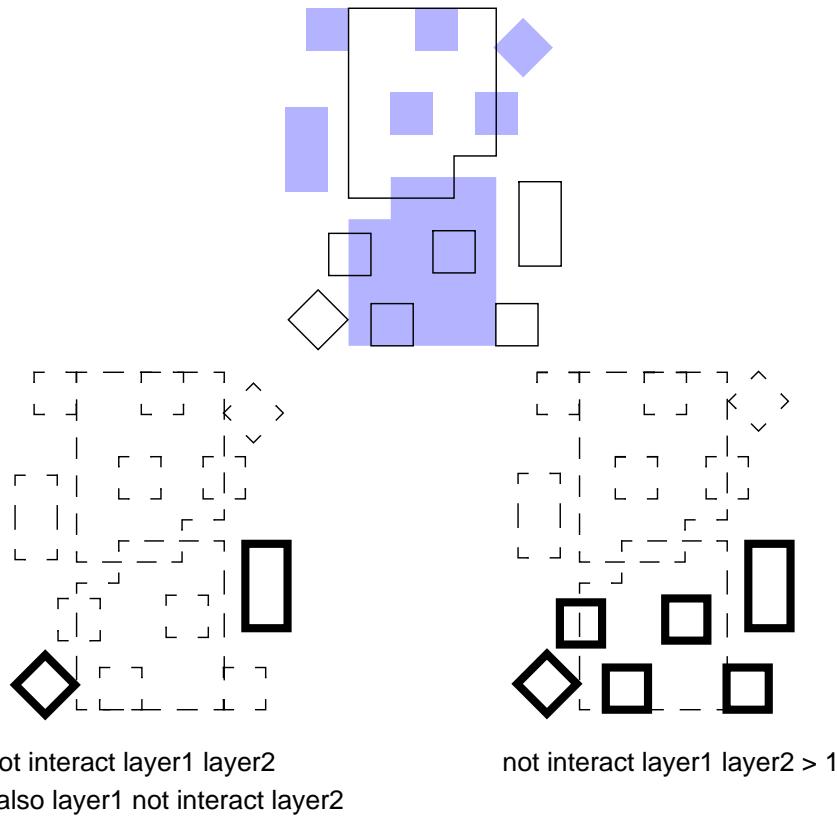
## Examples

### Example 1

Here are conceptual examples:

**Figure 4-187. Not Interact**

layer1      
layer2   



### Example 2

Using BY NET.

```
/* find well polygons that do not have wellties, or have wellties on more
than one net */
rule { well not interact welltie == 1 BY NET }
```

## Not Length

Layer operation

### NOT LENGTH *layer constraint*

#### Parameters

- *layer*

A required original layer or layer set, or a derived polygon or edge layer.

- *constraint*

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

The constraint must contain non-negative real numbers. It is interpreted in user units.

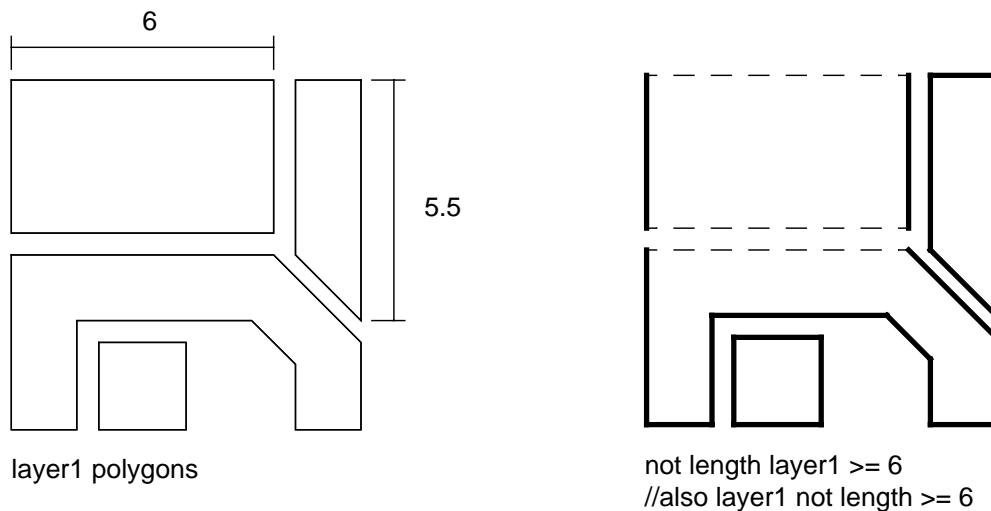
#### Description

Selects all *layer* edges that do not have length that meets the *constraint*. This is the complement of the [Length](#) operation. Length is in user units.

#### Examples

The Not Length operation shown in Figure 4-188 selects all layer1 edges that are not greater than or equal to 6 units.

**Figure 4-188. Not Length**



## Not Net

Layer operation

**NOT NET** *layer name* [*name* ...]

### Parameters

- *layer*

A required original layer or layer set, or a derived polygon layer.

- *name*

A required name of a net. You can specify *name* any number of times in one statement. The *name* can be a string variable (see [Variable](#)). The *name* can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters.

The string LVS\_POWER\_NAME may be used as a *name* argument. The string specifies all net names in a [LVS Power Name](#) statement. Similarly, the string LVS\_GROUND\_NAME specifies all net names in a [LVS Ground Name](#) statement. If either of these strings are specified, but the corresponding LVS specification statement is not, then the string is treated as a literal net name.

### Description

Selects all *layer* polygons that do not belong to nets having the specified *name*. This is the complement of the [Net](#) operation. To avoid ambiguity, you must use the parameters in the order shown above. Net name assignment is discussed under “[Net Name Specification](#)” and “[Label Attachment](#)” in the *Calibre Verification User’s Manual*.

The connectivity on *layer* must be established for the input layer.

The Not Net operation uses only top-level net names. The [Text Depth](#) statement controls which labels are used for top-level connectivity text.

By default, net names are case-insensitive; however the [Layout Preserve Case YES](#) specification statement layout causes the connectivity extractor to become fully case-sensitive and the Net operation will become case-sensitive as well.

See also [Connect](#), [Sconnect](#), [Stamp](#), [Net Area](#), [Net Interact](#), and [DRC Incremental Connect Warning](#).

### Examples

```
connect metall poly by contact
// derive a layer of all metall polygons that are not vdd
not_vdd_metal1 = NOT NET metall vdd VDD
// or not_vdd_metal1 = metall NOT NET vdd VDD
```

## Not Outside

Layer operation

**NOT OUTside *layer1* *layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.

### Description

Selects all *layer1* polygons having areas that are not completely outside *layer2* polygons. A *layer1* polygon that has coincident edges with *layer2* polygons, but does not share area, does not satisfy the Not Outside condition. This is the complement of the [Outside](#) operation. If either *layer1* or *layer2* is empty, there is no output.

See also [Inside](#), [Interact](#), [Not Inside](#), and [Not Outside Edge](#).

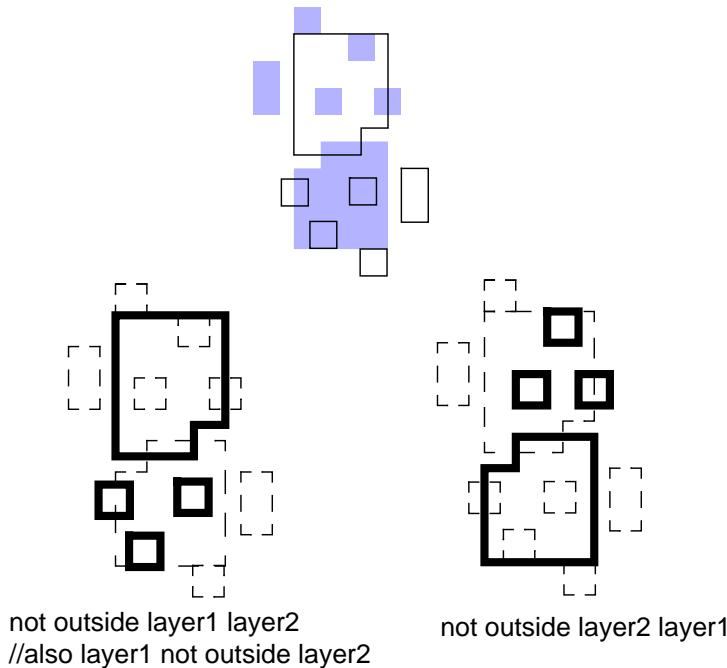
### Examples

Figure 4-189 shows two Not Outside operations. The operation on the left selects all *layer1* polygons that do not lie completely outside *layer2* polygons. The operation on the right reverses the layers and therefore selects *layer2* polygons that do not lie completely outside all *layer1* polygons.

**Figure 4-189. Not Outside**

layer1

layer2



## Not Outside Edge

Layer operation

**NOT OUTside EDGE** *layer1 layer2*

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.

### Description

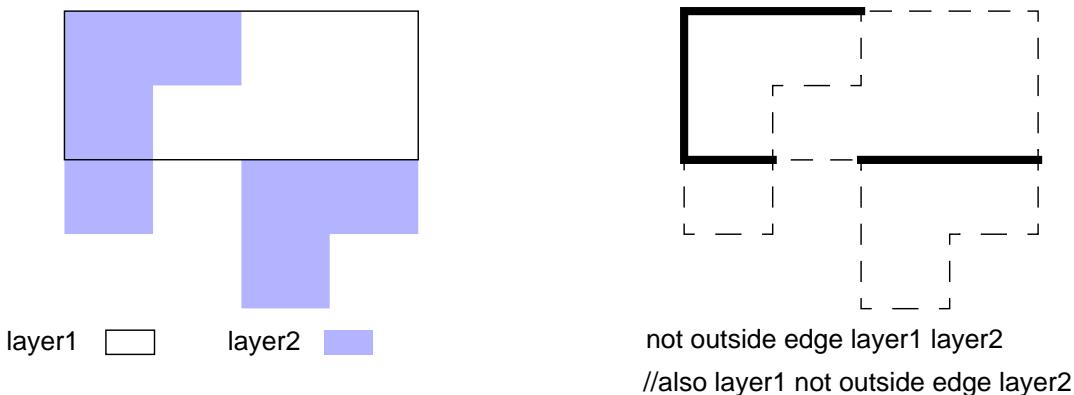
Selects all *layer1* edges or segments that do not lie completely outside *layer2* polygons. This operation is the complement of the [Outside Edge](#) operation.

See also [Inside Edge](#), [Coincident Edge](#), [Not Coincident Outside Edge](#), [Not Outside](#), and [Not Touch Outside Edge](#).

### Examples

The Not Outside Edge operation shown in Figure 4-190 selects all *layer1* edges or segments that do not lie completely outside *layer2* polygons or that are coincident with *layer2* edges.

**Figure 4-190. Not Outside Edge**



## Not Rectangle

Layer operation

**NOT RECTANGLE** *layer* [*constraint1* [BY *constraint2*]]  
 [ASPECT *constraint3*] [ORTHOGONAL ONLY | MEASURE EXTENTS]

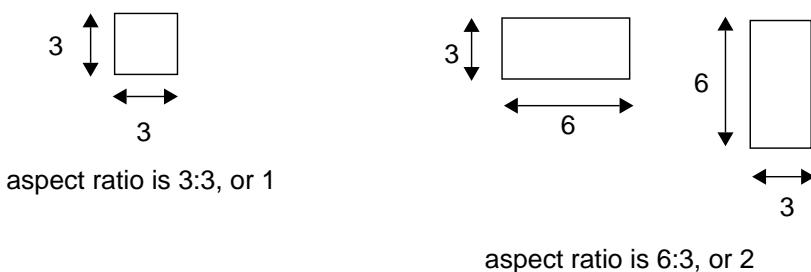
### Summary

Layer operation that selects polygons based on certain geometric properties they *do not* have.  
 This is the complement of the [Rectangle](#) operation.

### Parameters

- *layer*  
 An original layer or layer set, or a derived polygon layer.
- *constraint1*  
 An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.  
 The constraint must contain non-negative real numbers and is interpreted in user units.
- BY *constraint2*  
 An optional keyword set, where *constraint2* must follow the same guidelines as *constraint1*.  
 The optional keyword BY must always precede *constraint2*.
- ASPECT *constraint3*  
 An optional keyword and constraint that specifies the ratio of the longer side to the shorter side that a rectangle must not have in order to be output. [Figure 4-191](#) shows the aspect concept:

**Figure 4-191. ASPECT**



- ORTHOGONAL ONLY  
 An optional keyword that limits polygon selection to non-orthogonal (with respect to the database axes) rectangles. May not be specified with MEASURE EXTENTS.
- MEASURE EXTENTS  
 An optional keyword that selects polygons from the input layer having individual rectangular extents that do not meet the constraint. May not be specified with ORTHOGONAL ONLY.

## Description

Selects all non-rectangular polygons on *layer* if no constraint is specified. Can also select all *layer* rectangles with edge lengths not conforming to given constraints. If you specify only *constraint1*, the rectangles that are selected (in addition to other polygons) have at least one pair of edges that do not satisfy the constraint. If you specify *constraint1* BY *constraint2*, the rectangles that are selected have edge pairs that do not meet either constraint.

This is the complement of the [Rectangle](#) operation.

In the event that non-orthogonal (with respect to the coordinate axes) rectangles exist on *layer*, you should be careful when specifying constraints in the Not Rectangle operation. The problem arises when you make the assumption about the exact size of rectangles. For example, avoid using constraints such as “== 5” when *layer* contains non-orthogonal rectangles. A better choice of constraint might be “> 4.995 < 5.005”. This allows for a tolerance of ten database units, if your [Precision](#) is 1000.

If you specify ORTHOGONAL ONLY in a Not Rectangle operation, this outputs polygons such that:

- The dimensions do not satisfy the given constraints, if any, or
- The polygons’ edges are not parallel to the x and y axes, respectively. If the ORTHOGONAL ONLY keyword is omitted, then only the previous condition applies.

The MEASURE EXTENTS keyword is especially useful when attempting to select non-rectangular polygons, or rectangular polygons that are not orthogonal to the database axes, where the polygons do not fit within a specific rectangular extent. The dimensions of the rectangular extent are specified with a constraint.

## Examples

### Example 1

Basic contact check.

```
// Check that all contacts are rectangular with exact
// dimensions of .3 microns by .4 microns.

contact_size_error { not rectangle contact == .3 by == .4 }
```

### Example 2

Using ORTHOGONAL ONLY.

```
// Contacts must be .27 by .27 user units with edges
// parallel to the coordinate axes.

contact_rule { not rectangle contact == .27 by == .27
    orthogonal only }
```

## Not Rectangle

---

### Example 3

Using MEASURE EXTENTS.

```
// Find all vias having extents that do not fit inside a  
// .1 by .1 box  
  
bad_via { not rectangle via <= .1 by <= .1  
          measure extents }
```

### Example 4

Finding non-rectangular polygons.

```
// Generate a derived layer containing all contacts that are  
// not rectangular.  
  
not_rectangular_contacts = not rectangle contact
```

### Example 5

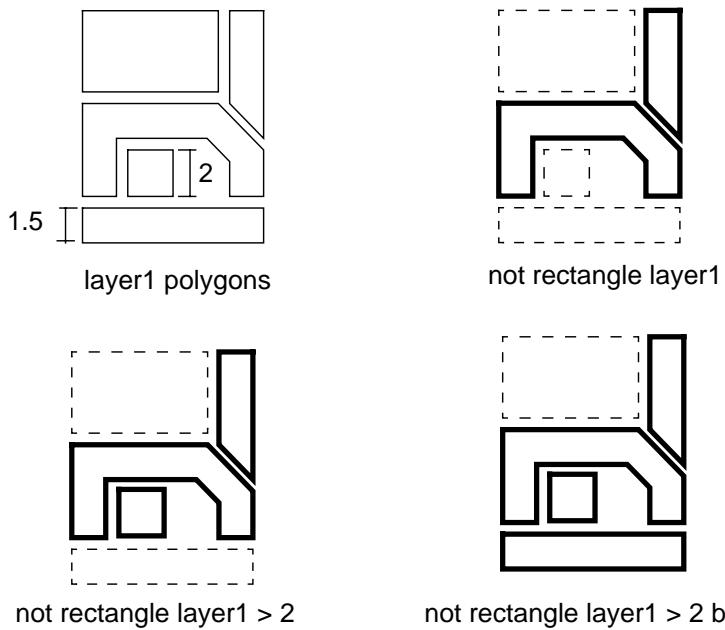
Using ASPECT.

```
/* Select contact rectangles that are not square. */  
  
rule { NOT RECTANGLE contact ASPECT == 1 }
```

### Example 6

Figure 4-192 shows three Not Rectangle operations. The operation on the top right selects layer1 polygons that are not rectangles. The operation on the bottom left selects layer1 polygons that are not rectangles or are rectangles and do not have a side greater than two user units. The operation on the bottom right selects layer1 polygons that are not rectangles or that are rectangles and do not have both sides greater than two user units.

**Figure 4-192. Not Rectangle**



## Not Touch

Layer operation

**NOT TOUCH** *layer1* *layer2* [*constraint* [BY NET] [EVEN | ODD]]

### Parameters

- *layer1*

A required original layer or layer set, or a derived polygon layer.

- *layer2*

A required original layer or layer set, or a derived polygon layer.

- *constraint*

A optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. The constraint specifies the number of *layer2* polygons a *layer1* polygon cannot touch in order to be selected.

- BY NET

An optional keyword used with the *constraint* parameter, which specifies the number of *layer2* polygons that are part of distinct nets, which a *layer1* polygon cannot touch in order to be selected. Touching is defined as sharing a coincident outside edge, or edge segment.

- EVEN

An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an even number. May not be used with ODD.

- ODD

An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is not an odd number. May not be used with EVEN.

### Description

Selects *layer1* polygons that do not share a coincident edge or segment as the only intersection with a *layer2* polygon. This is the complement of the [Touch](#) operation. If *layer2* is empty, the Not Touch operation generates output equivalent to the polygon data on *layer1*.

### BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that do not meet the constraint are output.

See also [Not Interact](#) and [Not Touch Edge](#).

## Not Touch

### Examples

#### Example 1

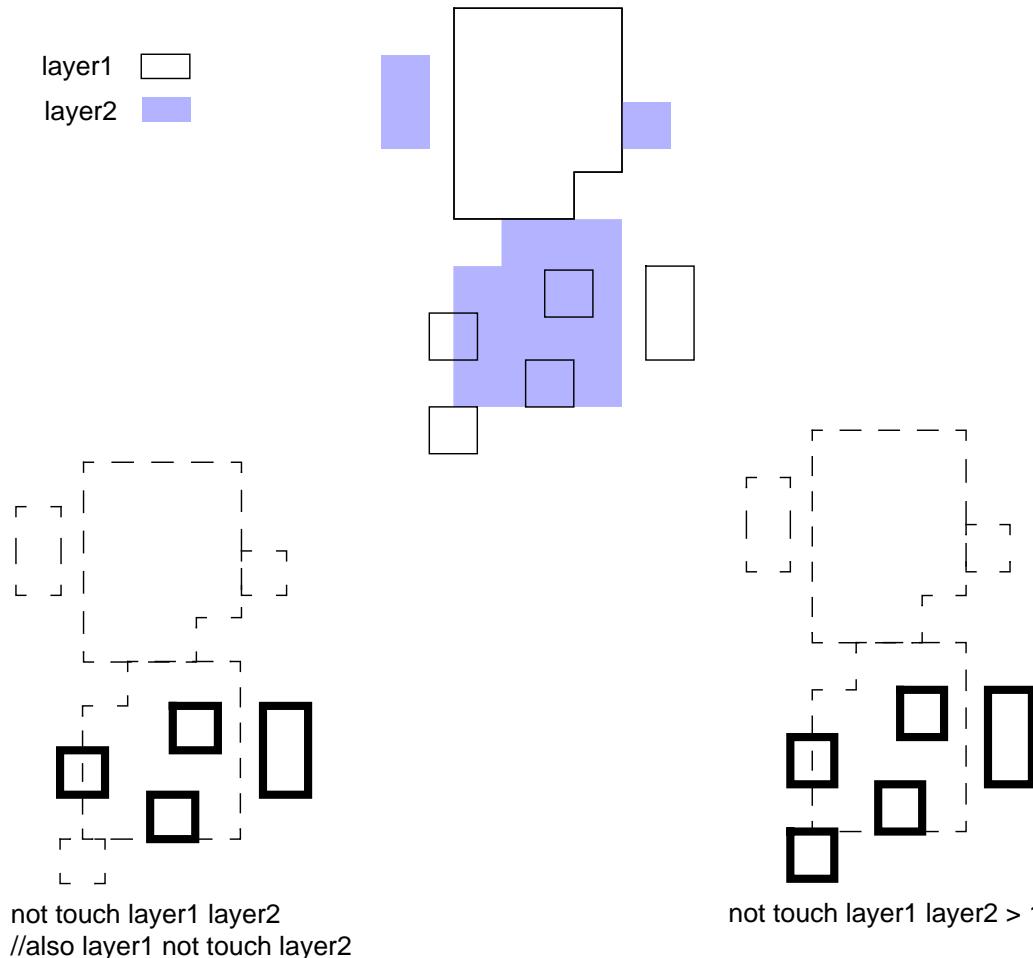
The following example uses the Not Touch operation to check for gates where polysilicon does not completely cross diffusion:

```
bad_gates {  
    @ "Gates" where poly does not completely cross diffusion.  
    gate = poly AND diff // Gate regions.  
    srcdn = diff NOT poly // Source/drain regions.  
    gate NOT TOUCH srcdn == 2  
    // Must touch two source/drain regions.  
}
```

#### Example 2

Figure 4-193 shows two Not Touch operations.

**Figure 4-193. Not Touch**



## Not Touch Edge

Layer operation

**NOT TOUCH EDGE** *layer1 layer2* [ENDPOINT {ALSO | ONLY}]

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.
- ENDPOINT {ALSO | ONLY}  
Optional keyword group that specifies edges are not selected from *layer1* that are collinear with and touch *layer2* edges only at their end points. ENDPOINT ALSO specifies not to select such *layer1* edges, which by default would be output. ENDPOINT ONLY specifies to select only the *layer1* edges that do not have this end point behavior.

### Description

Selects all complete *layer1* edges that are not coincident with any edges or edge segments on *layer2*. This is the complement of the [Touch Edge](#) operation. If *layer2* is empty, the Not Touch Edge operation generates output equivalent to the edge data on *layer1*.

By default, *layer1* edges that touch *layer2* edges only at their end points are output by this operation. The ENDPOINT ALSO and ENDPOINT ONLY options change this behavior.

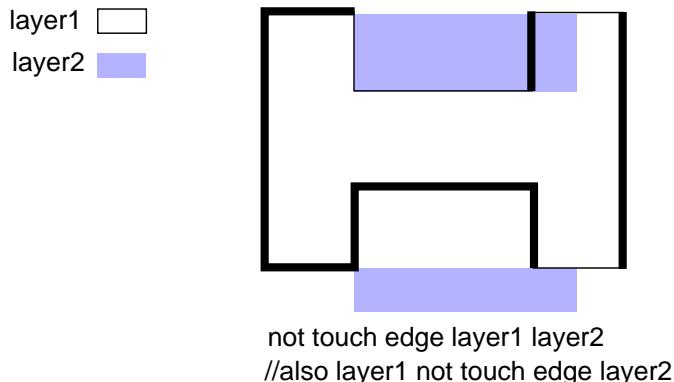
See also [Not Touch Inside Edge](#), [Not Touch Outside Edge](#), [Not Coincident Edge](#), and [Not Touch](#).

### Examples

#### Example 1

The Not Touch Edge operation shown in Figure 4-194 selects all *layer1* edges that are not coincident with a *layer2* edge or segment.

**Figure 4-194. Not Touch Edge**



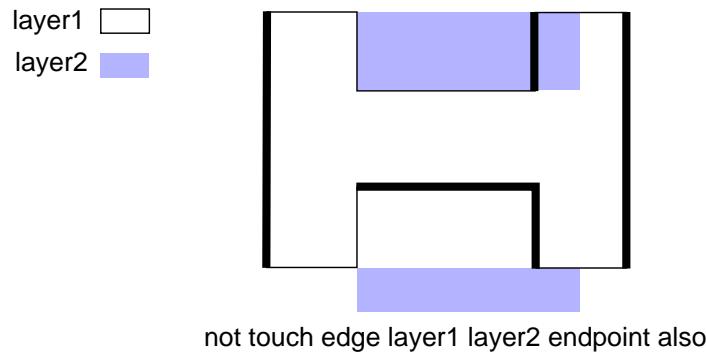
## Not Touch Edge

---

### Example 2

The Not Touch Edge ENDPOINT ALSO operation shown in Figure 4-195 selects all layer1 edges that are not coincident with a layer2 edge or segment, and layer1 edges that do not touch layer2 edges at their end points.

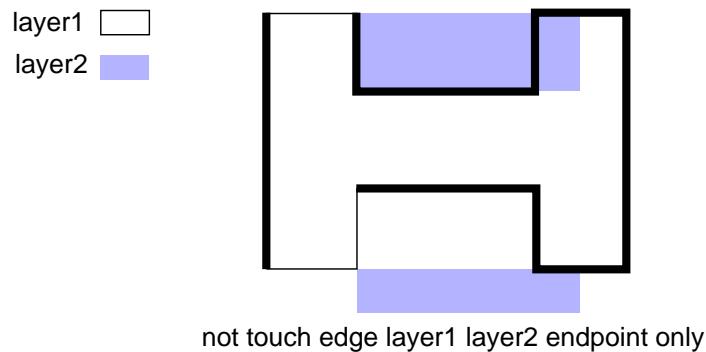
**Figure 4-195. Not Touch Edge ENDPOINT ALSO**



### Example 3

The Not Touch Edge ENDPOINT ONLY operation shown in Figure 4-196 selects all complete layer 1 edges that are not coincident with any edges or edge segments on layer2 and selects layer1 edges that do not touch layer2 edges at their end points.

**Figure 4-196. Not Touch Edge ENDPOINT ONLY**



## Not Touch Inside Edge

Layer operation

**NOT TOUCH INside EDGE *layer1* *layer2* [ENDPOINT {ALSO | ONLY}]**

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.
- ENDPOINT {ALSO | ONLY}  
Optional keyword group that specifies edges are not selected from *layer1* that are collinear with and touch *layer2* edges only at their end points. ENDPOINT ALSO specifies not to select such *layer1* edges, which by default would be output. ENDPOINT ONLY specifies to select only the *layer1* edges that do not have this end point behavior.

### Description

Selects all complete *layer1* edges that are not coincident with *layer2* edges, and including any coincident edges where there is no adjacent shared interior polygon area. This is the complement of the [Touch Inside Edge](#) operation. If *layer2* is empty, the Not Touch Inside Edge operation generates output equivalent to the edge data on *layer1*.

By default, *layer1* edges that touch *layer2* edges only at their end points are output by this operation. The ENDPOINT ALSO and ENDPOINT ONLY options change this behavior.

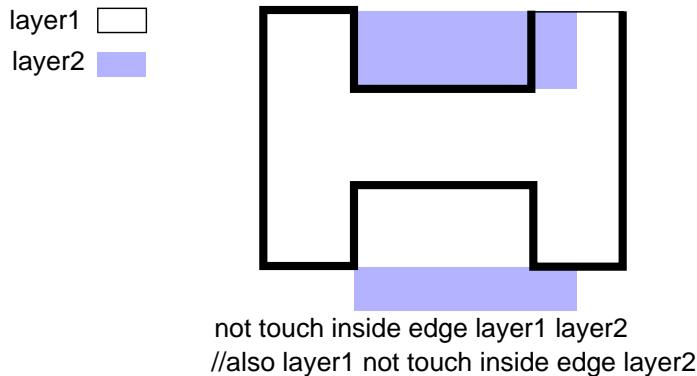
See also [Not Touch Edge](#), [Not Touch Outside Edge](#), [Not Coincident Inside Edge](#), and [Not Touch](#).

### Examples

#### Example 1

The Not Touch Inside Edge operation shown in Figure 4-197 selects the *layer1* edges that are not coincident with a *layer2* edge or segment.

**Figure 4-197. Not Touch Inside Edge**



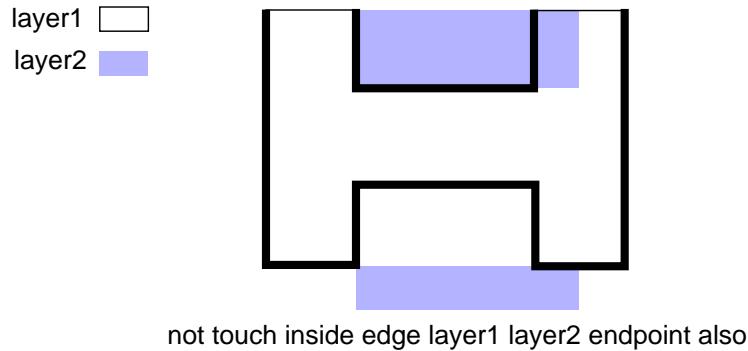
## Not Touch Inside Edge

---

### Example 2

The Not Touch Inside Edge ENDPOINT ALSO operation shown in Figure 4-198 selects the layer1 edges that are not coincident with a layer2 edge, and also selects layer1 edges that do not touch layer2 edges at their end points.

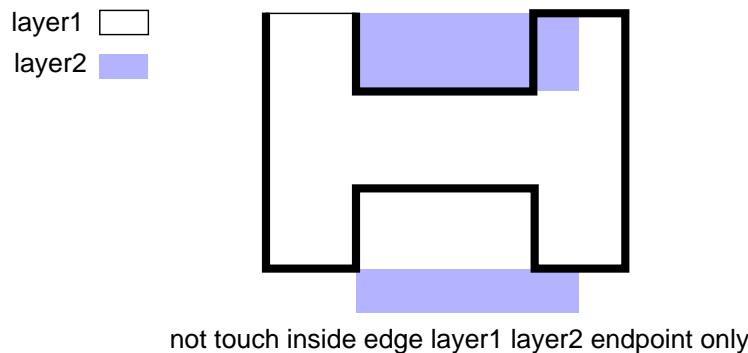
**Figure 4-198. Not Touch Inside Edge ENDPOINT ALSO**



### Example 3

The Not Touch Inside Edge ENDPOINT ONLY operation shown in Figure 4-199 selects only the layer1 edge segments that are not coincident with layer2 edges at their end points.

**Figure 4-199. Not Touch Inside Edge ENDPOINT ONLY**



## Not Touch Outside Edge

Layer operation

**NOT TOUCH OUTside EDGE** *layer1 layer2* [ENDPOINT {ALSO | ONLY}]

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.
- ENDPOINT {ALSO | ONLY}  
Optional keyword group that specifies edges are not selected from *layer1* that are collinear with and touch *layer2* edges only at their end points. ENDPOINT ALSO specifies not to select such *layer1* edges, which by default would be output. ENDPOINT ONLY specifies to select only the *layer1* edges that do not have this end point behavior.

### Description

Selects all complete *layer1* edges that are not coincident with *layer2* edges, and including any coincident edges where there is adjacent shared internal polygon area. This is the complement of the [Touch Outside Edge](#) operation. If *layer2* is empty, the Not Touch Outside Edge operation generates output equivalent to the edge data on *layer1*.

By default, *layer1* edges that touch *layer2* edges only at their end points are output by this operation. The ENDPOINT ALSO and ENDPOINT ONLY options change this behavior.

See also [Not Touch Edge](#), [Not Touch Inside Edge](#), [Not Coincident Outside Edge](#), and [Not Touch](#).

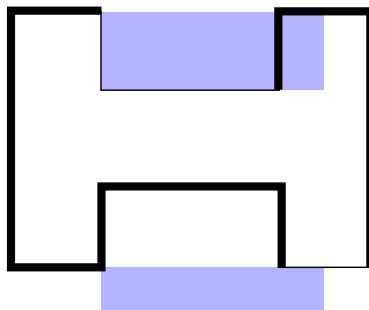
### Examples

#### Example 1

The Not Touch Outside Edge operation shown in Figure 4-200 selects the *layer1* edges that are not outside-coincident with a *layer2* edge or segment.

**Figure 4-200. Not Touch Outside Edge**

layer1   
layer2



not touch outside edge layer1 layer2  
//also layer1 not touch outside edge layer2

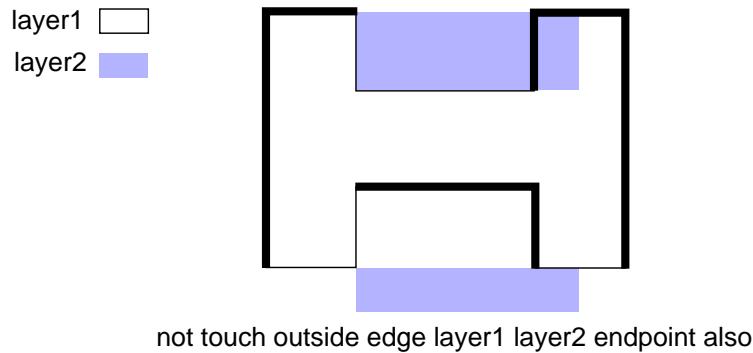
## Not Touch Outside Edge

---

### Example 2

The Not Touch Outside Edge ENDPOINT ALSO operation shown in Figure 4-201 selects the layer1 edges that are not outside-coincident with a layer2 edge or segment, and also selects layer1 edges that do not touch layer2 edges at their end points.

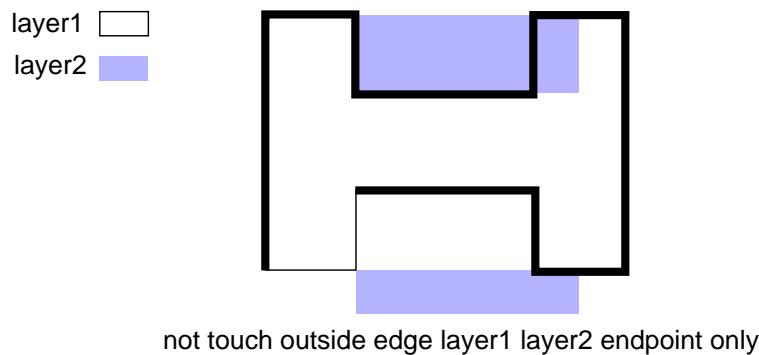
**Figure 4-201. Not Touch Outside Edge ENDPOINT ALSO**



### Example 3

The Not Touch Outside Edge ENDPOINT ONLY operation shown in Figure 4-202 selects the layer1 edges that are not outside-coincident with a layer2 edge or segment.

**Figure 4-202. Not Touch Outside Edge ENDPOINT ONLY**



## Not With Edge

Layer operation

**NOT WITH EDGE** *layer1 layer2 [constraint]*

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon layer.
- *layer2*  
A required derived edge layer.
- *constraint*  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. Specifies the number of edges or edge segments a *layer1* polygon must not have on *layer2* to be selected by the operation.

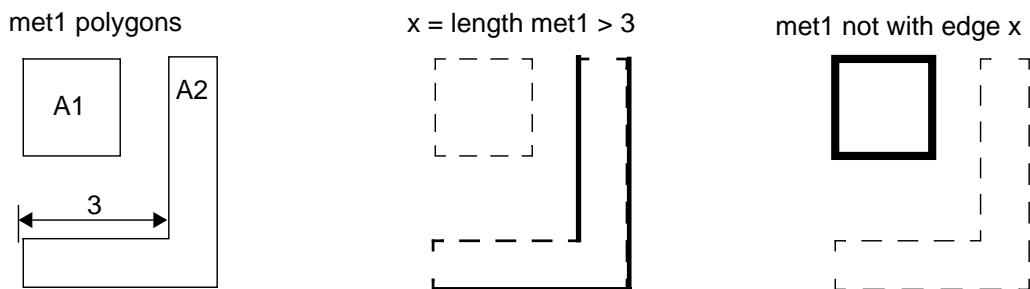
### Description

Selects all polygons on *layer1* that share no edges on *layer2*. This is the complement of the [With Edge](#) operation. If *layer2* is empty, the Not With Edge operation generates output equivalent to the polygon data on *layer1*.

### Examples

The Not With Edge operation shown in Figure 4-203 selects the met1 polygon that does not have edges of length greater than three.

**Figure 4-203. Not With Edge**



## Not With Neighbor

Layer operation

**NOT WITH NEIGHBOR** *layer1* [*layer2*] *constraint1* **SPACE** *constraint2*  
[SQUARE] [CENTERS [OCTAGONAL ONLY | ORTHOGONAL ONLY]]  
{ [INSIDE OF LAYER *layer3*] | [[NOT] CONNECTED]}

### Summary

Selects orthogonal (with respect to the database axes) rectangles if they do not have the specified number of orthogonal rectangles within the specified distance.

### Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- ***layer2***  
An optional original layer or layer set, or a derived polygon layer.
- ***constraint1***  
A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. Specifies the number of orthogonal (with respect to the database axes) rectangles a given orthogonal rectangle must not be in proximity to, in order to be output.
- **SPACE *constraint2***  
A required keyword and constraint containing non-negative, floating-point numbers in user units of length. The constraint may not be of the form “>”, “>=”, or “!=”. Specifies the dimensions of the measurement region around a *layer* polygon.
- **SQUARE**  
Optional keyword specifying the measurement region formed by *constraint2* uses the SQUARE metric rather than the Euclidean metric.
- **CENTERS**  
Optional keyword specifying the measurement region formed by *constraint2* is measured from the center of a given rectangle rather than from its perimeter. Distances are measured to the centers of other rectangles, rather than to their edges.
- **OCTAGONAL ONLY**  
Optional keyword that specifies the distance between the centerpoints of orthogonal rectangles is defined to satisfy the **SPACE** constraint only if the centerpoints are aligned at multiples of 45 degrees. May only be specified with the CENTERS keyword. May not be specified with the ORTHOGONAL ONLY keyword.

- ORTHOGONAL ONLY

Optional keyword that specifies the distance between the centerpoints of orthogonal rectangles is defined to satisfy the **SPACE** constraint only if the centerpoints are aligned in either the x- or y-direction. May only be specified with the CENTERS keyword. May not be specified with the OCTAGONAL ONLY keyword.

- INSIDE OF LAYER *layer3*

Optional keyword that restricts output to polygons inside of *layer3*. Refer to the algorithm description for more information. May not be specified with the CONNECTED keyword.

- [NOT] CONNECTED

Optional keyword that modifies the operation to consider only rectangles from *layer1* and *layer2*, which are on the same electrical nets (or different electrical nets if NOT is used). If this keyword is specified, the connectivity of *layer1* and *layer2*, if specified, are checked at compilation time. May not be specified with INSIDE OF LAYER.

## Description

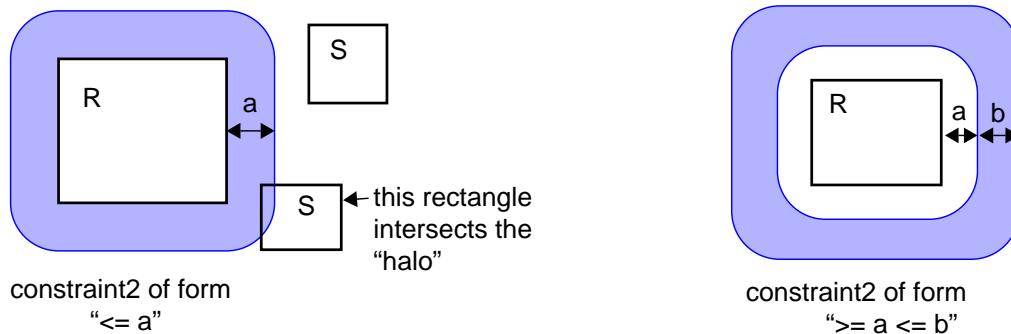
Selects orthogonal (with respect to the database axes) rectangles if they do not have the specified number of orthogonal rectangles within the specified distance. This is the complement of the **With Neighbor** operation.

With only *layer1* specified, the operation works as follows:

1. For each orthogonal rectangle, R, on the input layer, let C be the number of other orthogonal (with respect to the database axes) rectangles having a distance to R that satisfy **constraint2**.
2. If C does not satisfy **constraint1**, then R is output.

The distance between orthogonal rectangles R and S is determined using each edge of R and S in exactly the same manner as the **EXternal** operation. The Euclidean measurement metric is used. Measurement regions having a radius or radii determined by **constraint2** are constructed around each edge of R, creating a “halo” around R. The distance between R and S is said to satisfy **constraint2** if S intersects the halo. If R and S intersect at a singularity, then the distance is defined to be zero. See [Figure 4-344](#):

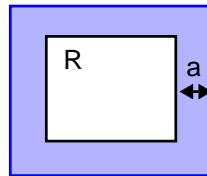
**Figure 4-204. Halo Region Using Euclidean Metric**



## Not With Neighbor

If **SQUARE** is specified, then the measurement region halo is constructed using the **SQUARE** metric rather than the Euclidean metric. See [Figure 4-345](#):

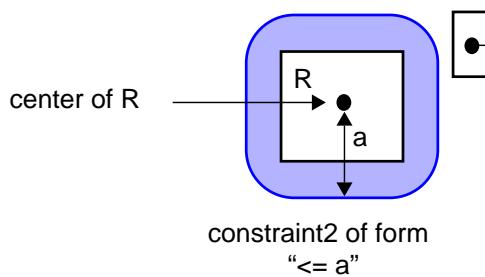
**Figure 4-205. Halo Region Using SQUARE Metric**



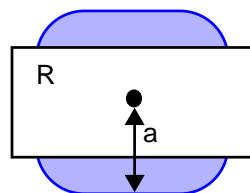
constraint2 of form  
“ $\leq a$ ”

If **CENTERS** is specified, then the measurement region halo is created about the center point of R. The distance between R and S satisfies **constraint2** if the center point of S intersects the halo. See [Figure 4-346](#):

**Figure 4-206. Measurement Region Using CENTERS**



The center of S must be within the halo region to satisfy the constraint.



When using **CENTERS**, it is possible the measurement region does not surround R, depending on the size of the constraint.

The **ORTHOGONAL ONLY** and **OCTAGONAL ONLY** keywords are used primarily for advanced pitch analysis and via alignment checks. These options are used with the **CENTERS** keyword and control how the alignment of centers is checked. **ORTHOGONAL ONLY** specifies alignment checking in the x- and y- directions. **OCTAGONAL ONLY** also does this, but also checks alignment diagonally at multiples of 45 degrees.

All With Neighbor and Not With Neighbor operations with the same input layer, **constraint2**, and **SQUARE** and **CENTERS** keyword specifications are executed concurrently.

If *layer2* is specified, then the algorithm changes (slightly) as follows:

1. For each orthogonal rectangle R on *layer1*, let C be the number of orthogonal rectangles on *layer2* whose distance to R that satisfy *constraint2*.
2. If C does not satisfy *constraint1*, then R is output.

The distance between two rectangles from *layer1* and *layer2* is defined to be 0 if the rectangles touch at any point, except when CENTERS is specified. If CENTERS is specified, then the distance between rectangles on *layer1* and *layer2* is measured between their centers, regardless of any overlap.

If INSIDE OF LAYER *layer3* is specified, then the algorithm is as follows:

For each polygon P on *layer3*,

1. For each orthogonal rectangle R on *layer1*, which is inside of P,
  - o If only *layer1* is specified, let C be the number of other orthogonal rectangles on *layer1*, which are also inside of P and whose distance to R satisfies *constraint2*, or
  - o If *layer2* is specified, let C be the number of orthogonal rectangles on *layer2*, which are also inside of P and whose distance to R satisfies *constraint2*.
2. If C does not satisfy *constraint1*, then R is output.

If [NOT] CONNECTED is specified, then the algorithm is as follows:

1. For each orthogonal rectangle R on *layer1*,
  - o If only *layer1* is specified, let C be the number of other orthogonal rectangles on *layer1* such that the rectangles are on the same (or different, if NOT is used) electrical net as R, and whose distance to R satisfies *constraint2*, or
  - o If *layer2* is specified, let C be the number of orthogonal rectangles on *layer2* such that the rectangles are on the same (or different, if NOT is used) electrical net as R, and whose distance to R satisfies *constraint2*.
2. If C does not satisfy *constraint1*, then R is output.

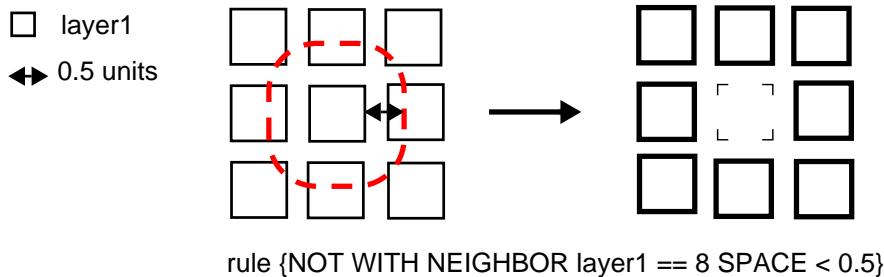
## Not With Neighbor

---

### Examples

#### Example 1

**Figure 4-207. Example of Not With Neighbor**



#### Example 2

To find isolated contacts, any polygon on the contact layer that does not have at least one neighboring contact within maximum contact spacing is considered isolated:

```
iso_contact {
    NOT WITH NEIGHBOR cont >= 1 SPACE <= max_cont_space
}
```

## Not With Text

Layer operation

**NOT WITH TEXT** *layer name* [*text\_layer*] [PRIMARY ONLY] [CASE SENSITIVE]

### Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- ***name***  
A required name of a free-standing text object. The ***name*** can be a string variable (see [Variable](#)). The ***name*** can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters.
- ***text\_layer***  
An optional original layer or layer set.
- **PRIMARY ONLY**  
An optional keyword specifying that only top-cell text is to be used in the operation.
- **CASE SENSITIVE**  
An optional keyword that specifies ***name*** parameters must match by case.

### Description

Selects all ***layer*** polygons that do not intersect the position of a text object having the specified ***name***. This is the complement of the [With Text](#) layer operation.

If *text\_layer* is specified, only polygons not having the text objects on *text\_layer* are selected. The ***name*** parameter is case-insensitive by default. The parameter order given must be observed to avoid ambiguity.

[Layout Text](#) and [Layout Text File](#) text objects participate in Not With Text operations.

[Text](#), [Text Depth](#), and [Text Layer](#) statements have no effect in Not With Text operations.

### Examples

```
// Select all large pads that do not have a test-pads label:  
large_pads = pads AREA > 5000  
test_pads = large_pads NOT WITH TEXT "test-pads" 14
```

## Not With Width

Layer operation

### NOT WITH WIDTH *layer constraint*

#### Parameters

- *layer*  
An original layer, layer set, or a derived polygon layer.
- *constraint*  
A required constraint interpreted as width in user units. It is one of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

#### Description

Selects regions of *layer* polygons that do not satisfy the width *constraint*. This operation does not select entire input polygons unless an entire polygon does not meet the *constraint*. Behaves as a non-node-preserving layer constructor. This is the complement of the [With Width](#) operation (which see for a complete discussion of the algorithms used for determining width).

The concept of width is somewhat difficult to define in a consistent way. This operation uses a consistent width definition and replaces complex SVRF derivations used in determining width.

In the case of angled geometry, the output conforms to intuitive notions of width. This output may be contrary (and more useful) to what you would see using [Size](#), especially for layers that have received OPC corrections.

Not With Width operations using the same layer are performed concurrently when the layer data is orthogonal to the database axes.

See also [Internal](#).

#### Examples

```
// derive polysilicon regions having width > .10
wide_poly = poly not with width <= .10

// derive input polysilicon polygons having width > .10
wide_poly = poly not with width <= .10
orig_wide_poly = wide_poly interact poly
```

# Offgrid

Layer operation

**OFFGRID** *layer {resolution | x-resolution y-resolution}*  
 [INSIDE OF LAYER *layer2* [RELATIVE | ABSOLUTE]]  
 [CENTERS] [EDGE] [REGION *[value]*]  
 [[OFFSET *[x\_offset y\_offset]*] [INSIDE BY *value* | OUTSIDE BY *value*]] ...]  
 [HINT]

## Summary

Layer operation that outputs offgrid objects based upon a user-specified grid. Output is a derived error layer by default, but can be derived polygons or edges. This operation is very useful for advanced pitch analysis.

## Parameters

- ***layer***  
 A required original polygon layer, derived polygon layer, or derived edge layer. The ***layer*** and ***resolution*** parameters should be ordered as shown in the syntax description to avoid ambiguity.
- ***resolution | x-resolution y-resolution***  
 Required positive integer(s) in database units that specify the grid resolution. The value of ***resolution*** is applied in both the x and y directions. If ***x-resolution*** and ***y-resolution*** are specified, ***x-resolution*** is applied in the x direction and ***y-resolution*** in the y direction.
- INSIDE OF LAYER *layer2* [RELATIVE | ABSOLUTE]  
 An optional keyword set that specifies only objects from the input ***layer*** that lie inside *layer2* are used by the operation. The *layer2* parameter must be an original or derived polygon layer. Two additional keywords may be specified with INSIDE OF LAYER:
  - RELATIVE — Specifies that the grid origin is determined by the total rectangular extent of ***layer*** objects inside of *layer2*. This is the default when INSIDE OF LAYER is specified.
  - ABSOLUTE — Specifies that the grid origin is determined by the unmerged extents of polygons from *layer2*.

This is discussed under [INSIDE OF LAYER](#).

- CENTERS  
 Optional keyword that specifies the centers of input objects are checked against the grid. If the input ***layer*** is a polygon layer, then the centers of the extents of input polygons are checked. If the input ***layer*** is a derived edge layer, then the midpoints of input edges are checked.

- **EDGE**

An optional keyword that specifies the output of the operation is a derived edge layer. This keyword may not be specified with CENTERS if the input **layer** is a polygon layer.

- **REGION [value]**

An optional keyword that specifies the output is a derived polygon layer consisting of squares that mark the locations of offgrid vertices. By default, the squares are  $1 \times 1$  user units and have centers at the coordinates of offgrid vertices. The optional *value* is a positive floating-point number in user units that defines the edge length of the squares. This keyword may not be used if the input **layer** is a derived edge layer.

- **OFFSET [x\_offset y\_offset] [INSIDE BY value | OUTSIDE BY value]**

An optional keyword and value set that indicates an offset displacement to apply to objects before checking them against the **resolution** parameters. One or more OFFSET parameter groups may be specified. Each OFFSET specification is applied in the order it appears in the operation.

- *x\_offset y\_offset* — These parameters are any integers in database units. Negative integers must be enclosed in parentheses ( ). The *x\_offset* is applied in the x-direction, and the *y\_offset* is applied in the y-direction. If the input layer is a derived edge layer, then the tool takes the absolute values of the *x\_offset* and *y\_offset* parameters before applying them. This parameter group may be specified with either the INSIDE BY or OUTSIDE BY option, which provide directional offsets. If this occurs, then both the absolute offsets and the directional offset apply.
- **INSIDE BY value** — Specifies an offset displacement to apply to edges before checking them. This option may only be used when the input **layer** is a derived edge layer. The *value* is a positive integer in database units. Each input edge is offset in the inside direction (as compared to the polygon from which the edge originates) by *value* database units before being checked. May be specified with *x\_offset y\_offset*, in which case both conditions apply. May not be specified with OUTSIDE BY in the same OFFSET keyword group.
- **OUTSIDE BY value** — Specifies an offset to apply to edges before checking them. This option may only be used when the input **layer** is a derived edge layer. The *value* is a positive integer in database units. Each input edge is offset in the outside direction (as compared to the polygon from which the edge originates) by *value* database units before being checked. May be specified with *x\_offset y\_offset*, in which case both conditions apply. May not be specified with INSIDE BY in the same OFFSET keyword group.

- **HINT**

An optional keyword that specifies the output objects provide locations of nearby on-grid points. This keyword may not be specified with either EDGE or REGION.

## Description

Generates a derived error layer by flagging all off-grid objects on the input layer. These objects include polygon vertices, edges, or polygon centers depending on the specified keywords. A point is off-grid if its coordinates are not proper multiples of the resolution values.

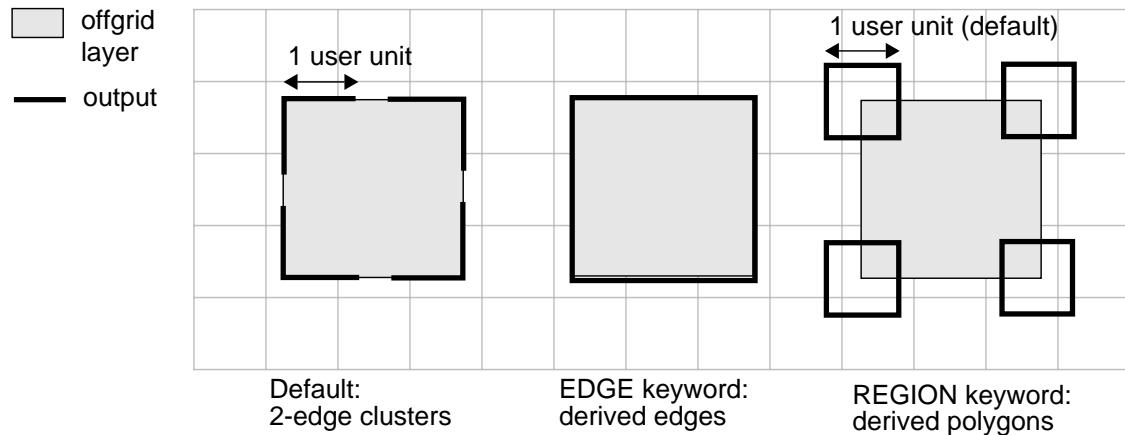
The grid used for checking objects is defined by either the ***resolution*** or ***x-resolution*** and ***y-resolution*** parameters. For example, if you specify the ***resolution*** as 45, then all relevant points (vertices, endpoints, or center points) of objects are checked to see if their x and y coordinates are on a 45 nm grid (database units). The order of the ***resolution*** and ***layer*** parameters is important to avoid ambiguity. These resolution parameters have no interaction with the **Resolution** statement.

By default, this operation is error-directed. The REGION keyword specifies derived polygon output; the EDGE keyword specifies derived edge output.

If the ***layer*** is a polygon layer, then a polygon P is checked by the operation as follows:

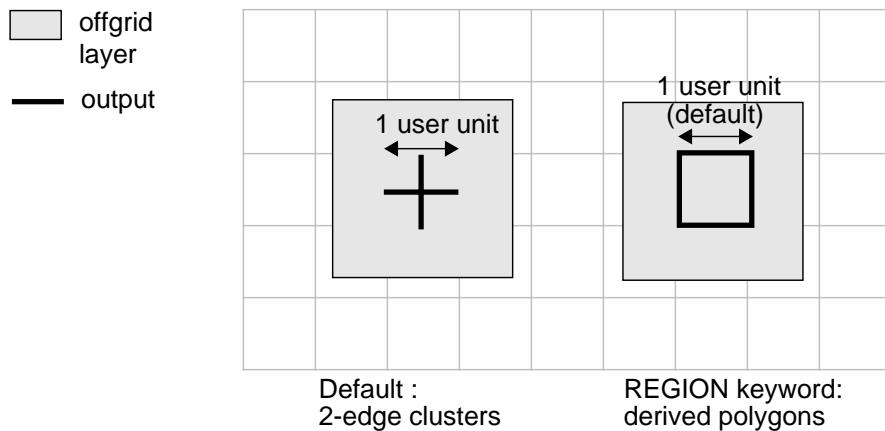
1. If CENTERS is not specified, then every vertex (x,y) on P is checked against the resolution parameters. If the vertex (x,y) is offgrid, then output is according to the following rules:
  - o If output is a derived error layer (the default), then a two-edge cluster is output: the two edges are adjacent edges of the geometry whose common endpoint is (x,y). For clarity in results presentation, these edges are trimmed to a maximum distance of one user unit from the location of the offgrid vertex.
  - o If output is a derived edge layer (EDGE is specified), then two edges are output. The two edges are adjacent edges of the geometry whose common endpoint is (x,y).
  - o If output is a derived polygon layer (REGION *value* is specified), then a *value* × *value* user unit square whose center point is (x,y) is output. If *value* is not specified, then the squares have edges one user unit long.

**Figure 4-208. Output Without CENTERS**



2. If CENTERS is specified, then the center point (cx,cy) of P's extent is checked against the resolution parameters. CENTERS may not be specified with EDGE for polygon input layers. If the center point is offgrid, then output is according to the following rules:
  - o If output is a derived error layer, then a two-edge cluster is output where one edge is vertical and one edge is horizontal. The edges are trimmed so as not to exceed the extent of the checked polygon. If trimming is unnecessary, then the edges are each one user unit long. The output edges intersect at their midpoints, and the intersection point coincides with (cx,cy).
  - o If output is a derived polygon layer (REGION *value* is specified), then a *value* × *value* square whose center point is (cx,cy) is output. If *value* is not specified, then the squares have edges one user unit long.

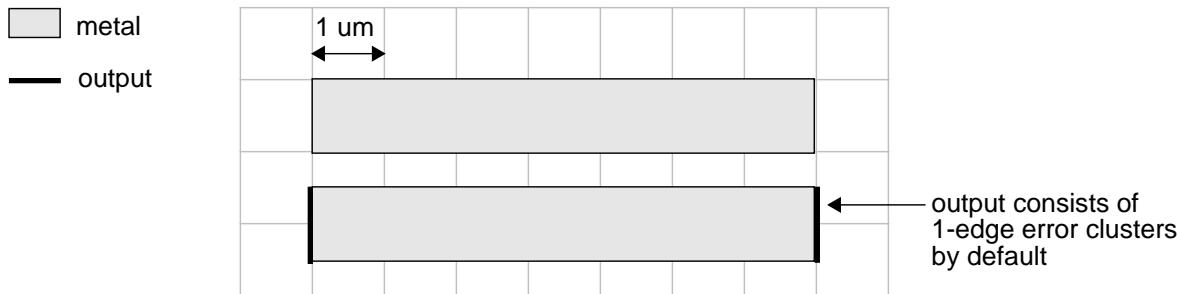
**Figure 4-209. CENTERS With Polygon Input**



If the input *layer* is a derived edge layer, then an edge E is checked by the operation as follows.

1. If CENTERS is not specified, then the endpoints of E are checked against the resolution parameters. Edge E is offgrid if either endpoint fails, and output is according to the following rules:
  - o If output is a derived error layer (the default), then E is output as a one-edge error cluster.
  - o If output is a derived edge layer (EDGE is specified), then E is output.

See [Figure 4-210](#).

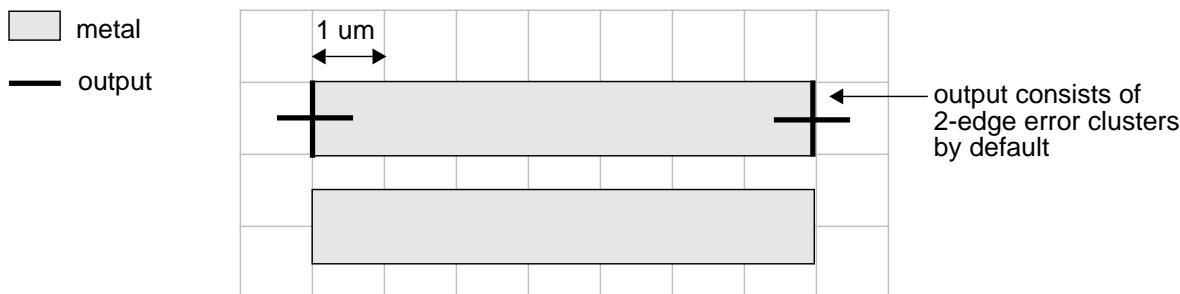
**Figure 4-210. Edge Input Without CENTERS**

```
big_bus_edge = LENGTH metal == 1
offgrid_bus_edge { OFFGRID big_bus_edge 1000 }
// if EDGE is specified, output is a derived edge layer
```

2. If CENTERS is specified, then the midpoint ( $mx, my$ ) of E is checked against the resolution parameters. If the midpoint is offgrid, then output is according to the following rules:

- o If output is a derived error layer, then a two-edge cluster is output where one edge is vertical and one edge is horizontal. The output edges are trimmed so as not to exceed the length of the derived edge. If trimming is unnecessary, then the edges are each one user unit long. The edges intersect at their midpoints, and the intersection point coincides with ( $mx, my$ ).
- o If output is a derived edge layer (EDGE is specified), then E is output.

See [Figure 4-211](#).

**Figure 4-211. Edge Input With CENTERS**

```
big_bus_edge = LENGTH metal == 1
offgrid_bus_edge { OFFGRID big_bus_edge 1000 CENTERS}
// if EDGE is specified, output is a derived edge layer
```

## INSIDE OF LAYER

If INSIDE OF LAYER is specified, then only objects on the input *layer* that lie inside *layer2* polygons are used by the operation.

The behavior is as follows:

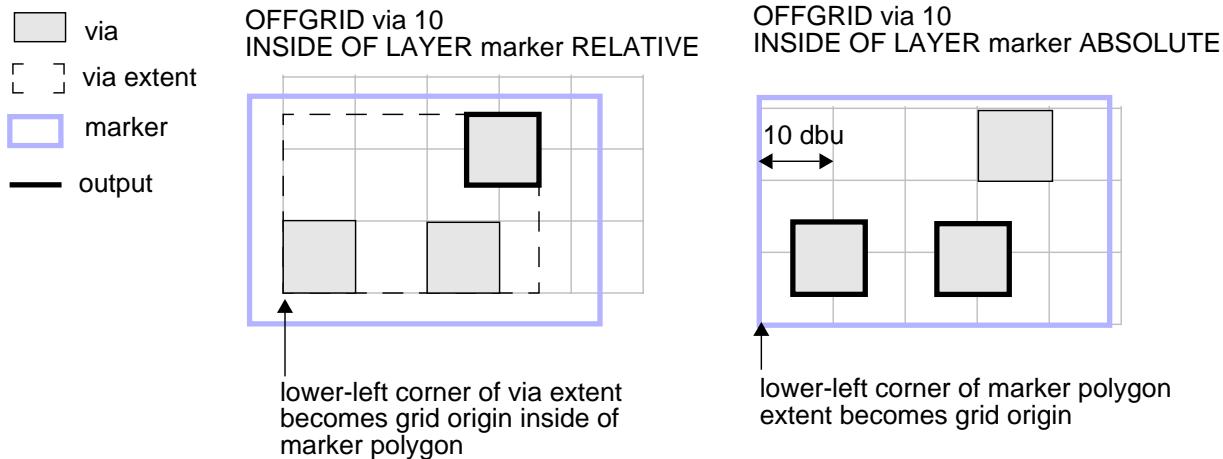
- If *layer* is polygon-type, then only *layer* polygons inside (see [Inside](#) for a definition) of *layer2* polygons participate in the checking process.
- If *layer* is edge-type, then only *layer* edges with no portion outside of polygons on *layer2* participate in the checking process. See [Outside](#) for a definition, but remember the Outside operation does not use edge layer input.

The default option for determining the grid is RELATIVE. This means the rectangular extent of *layer* objects inside of each *layer2* polygon is calculated. The lower-left corner of the extent is used as the grid origin reference for the *layer* objects inside the *layer2* polygon.

The ABSOLUTE option causes the total rectangular extent of each *layer2* polygon to be calculated. The lower-left corner of this extent is used as the grid origin reference for the *layer* objects inside the *layer2* polygon.

Specifying INSIDE OF LAYER is useful for advanced pitch checks.

**Figure 4-212. INSIDE OF LAYER With Polygon Input**



## OFFSET

The OFFSET keyword group allows an offset to be applied to objects prior to checking against the specified grid. Regardless of any specified offset, output is in terms of the original location of objects that are read in.

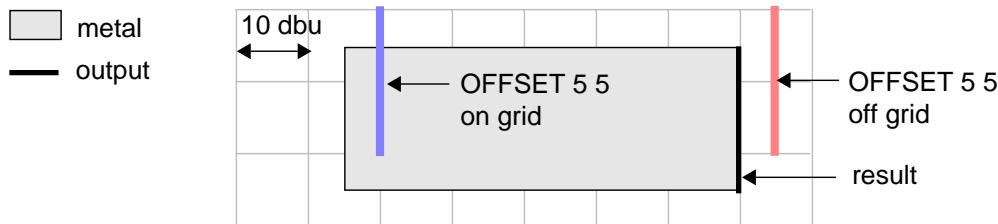
The INSIDE BY and OUTSIDE BY options apply only to derived edge input layers.

More than one OFFSET parameter group may be specified in an operation. If more than one is specified, they are applied one-by-one in the order they are specified.

The presence of any offset keyword group modifies the operation's behavior as follows:

- If *layer* is polygon-type, then for each vertex or center point (when CENTERS is specified) to be checked, apply each OFFSET keyword group. This means the point is shifted by the OFFSET *x\_offset y\_offset* parameters and then checked to see if the shifted point is on grid. If the point is on grid after any offset is applied, then there is no output.
- If *layer* is edge-type, then for each edge E to check, apply each OFFSET keyword group. This means to shift E by the absolute values of *x\_offset* and *y\_offset* when specified. If INSIDE BY or OUTSIDE BY are specified, then E is shifted by the specified *value* and toward the inside or outside of the polygon from which E originates. If CENTERS is specified, then the midpoint of E is computed after this shift. If the edge or midpoint is on grid after any OFFSET is applied, then there is no output.

**Figure 4-213. OFFSET With Edge Input**



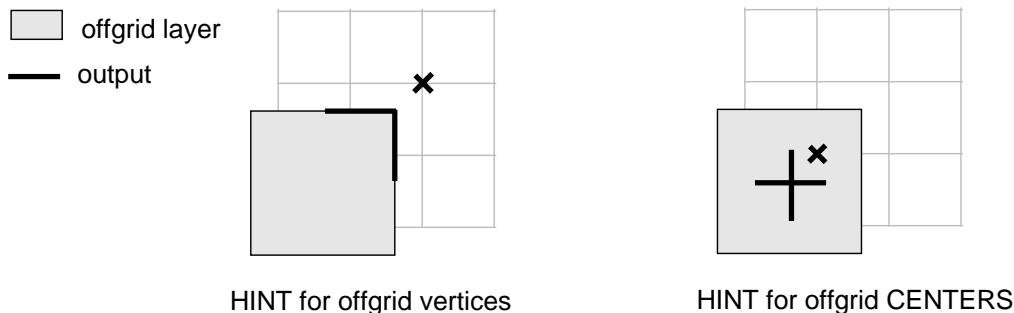
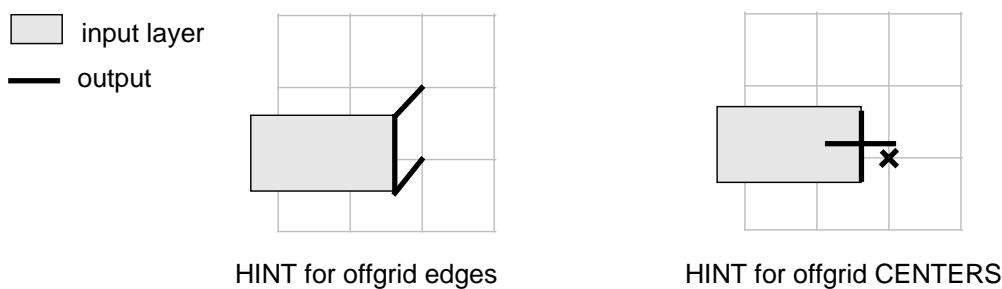
```
m_line_end = CONVEX EDGE metal == 2 WITH LENGTH == 0.20
// metal line ends must be on a 5 dbu grid, odd multiples only
offgrid_line_end { OFFGRID m_line_end 10 OFFSET 5 5 }
```

### HINT

The HINT keyword provides locations of nearby on-grid points. These can be useful for correcting the locations of off-grid objects. The locations of the HINT markers are simply suggestions for on-grid locations. This keyword only uses error-directed output, so the EDGE and REGION keywords may not be specified with HINT. The HINT output is as follows:

- If the *layer* is a polygon layer and CENTERS is not specified (that is, each vertex is being checked), then the output for each offgrid vertex is a 4-edge cluster. The first two edges are original edges of the polygon, with a maximum length of one user unit, which straddle the offgrid vertex. The third and fourth edges form an “X” consisting of two 45-degree edges whose intersection point is a nearby on-grid point. The extent of the “X” is approximately one-tenth that of the first two edges in the cluster.
- If the *layer* is a derived edge layer and CENTERS is not specified (that is, each endpoint is being checked), then the output for each offgrid edge consists of a 2- or 3-edge cluster. The first edge in the cluster is the actual offgrid edge itself. Then for each offgrid endpoint, an edge is drawn connecting it to a nearby on-grid point and that edge becomes an element of the output cluster.
- If CENTERS is specified (that is, polygon or edge center-points are being checked), then the output for each offgrid center point is a 4-edge cluster. The first two edges form

a “+” shape whose intersection is the offgrid center point. The third and fourth edges form an “X” consisting of two 45-degree edges whose intersection point is a nearby on-grid point. The extent of the “X” is approximately one-tenth that of the first two edges in the cluster.

**Figure 4-214. Polygon Input With HINT****Figure 4-215. Edge Input With HINT**

## Examples

### Example 1

The following rule check implements the specification “Diffusion must be on a 0.025 micron grid (assuming 1000 database units per user-unit)”:

```
rule { OFFGRID diffusion 25 }
```

### Example 2

This example shows a special case. The following rule check checks metal to a two-database-unit resolution grid because 3 -1 is evaluated as 2:

```
rule {  
OFFGRID metal 3 -1}
```

whereas the following example results in a compilation error due to the negative y value for the snap grid:

```
rule {  
OFFGRID metal 3 (-1)}
```

### Example 3

In this rule, all polygons on layer POLY with width exactly 0.06 must have horizontal or vertical pitch (centerline to centerline spacing) on a grid that is a multiple of 250 nm. This measurement is with respect to each polygon on layer POLY\_OPC:

```
rule {
    poly_h = ANGLE poly == 0
    poly_v = ANGLE poly == 90

    // Regions of exact horizontal or vertical width:
    poly_6h = INT poly_h == 0.06 REGION OPPOSITE
    poly_6v = INT poly_v == 0.06 REGION OPPOSITE

    // Offset the poly regions by 0.3 um
    poly_6h_shift = SHIFT poly_6h BY 0 0.3
    poly_6v_shift = SHIFT poly_6v BY 0.3 0

    // Centerlines of these offset regions:
    poly_6h_centerline = poly_6h_shift INSIDE EDGE poly_6h
    poly_6v_centerline = poly_6v_shift INSIDE EDGE poly_6v

    // Centerlines are on a 250 dbu grid, horizontally and
    // vertically, respectively.
    OFFGRID poly_6h_centerline 1 250 INSIDE OF LAYER poly_opc
    OFFGRID poly_6v_centerline 250 1 INSIDE OF LAYER poly_opc
}
```

### Restricted Design Rule Examples

For the following examples, the following rule file settings apply.

```
VARIABLE xpitch 0.21 // contact pitch in x-direction.
VARIABLE ypitch 0.28 // contact pitch in y-direction.
VARIABLE optrad 2    // multiplier for optical radius influence.
VARIABLE halfcont 0.06 // half the width of the contacts.
```

### Example 3

This is a simpler type of contact check. Centers of contacts must be on grid. Extents of a marker layer are used to determine the grid. Hints are provided for nearby on-grid locations.

```
offgridcont { OFFGRID cont xpitch*1000 ypitch*1000
               INSIDE OF LAYER gridmarker ABSOLUTE CENTERS HINT }
```

### Example 4

Contacts within an optical radius of each other need to be on grid relative to each other. Centers are checked rather than edges. Pitches are unequal in the x- and y-directions, which is why Grow is used instead of Size.

```
// Derive contact groups lying within an optical radius.
// Perimeters need to coincide with contact centers, not edges.
cogroup = EXTENTS ( GROW cont RIGHT BY xpitch*optrad-halfcont
                     LEFT BY xpitch*optrad-halfcont
                     TOP BY ypitch*optrad-halfcont
                     BOTTOM BY ypitch*optrad-halfcont )
```

## Offgrid

---

```
// Find offgrid contact centers within cogroup polygons.  
// Grid is referenced to lower-left corners of cogroup polygons.  
// Provide hints to on-grid locations.  
offgridcont = OFFGRID cont xpitch*1000 ypitch*1000  
    INSIDE OF LAYER cogroup ABSOLUTE CENTERS HINT  
  
// Collect the results in an RDB.  
contact_pitch { DFM RDB offgridcont "contactpitch.rdb"  
    ALL CELLS NOPSEUDO CHECKNAME offgridcont }
```

### Example 5

This check ensures that metal1 line ends enclosing contacts by no more than 0.2 um enclose the contacts by a 90 nm offset from the contact grid. The gridmarker layer is used to set the extents of the grid.

```
// derive horizontal and vertical m1 line ends enclosing contacts  
// by no more than 0.2 um.  
m1lineend = m1 CONVEX EDGE == 2 WITH LENGTH == 0.16  
mlencedge = ENC cont [m1lineend] < 0.2  
mlencedgeh = mlencedge ANGLE == 0  
mlencedgev = mlencedge ANGLE == 90  
  
// check that the m1 edge centers have 90 dbu offset from contacts  
// and are on pitch. a resolution of 1 dbu means the edge can  
// appear anywhere along that axis.  
// horizontal edges must agree with y-pitch.  
offgridmlh = OFFGRID mlencedgeh 1 ypitch*1000 CENTERS  
    INSIDE OF LAYER gridmarker ABSOLUTE  
    OFFSET INSIDE BY 90 HINT  
// vertical edges must agree with x-pitch.  
offgridmlv = OFFGRID mlencedgev xpitch*1000 1 CENTERS  
    INSIDE OF LAYER gridmarker ABSOLUTE  
    OFFSET INSIDE BY 90 HINT  
  
// collect the output and report it.  
offgridml = DFM COPY offgridmlh offgridmlv  
offgridml { DFM RDB offgridml "m1grid.rdb" ALL CELLS NOPSEUDO  
    CHECKNAME offgridml }
```

## Opcbias

Layer operation

```
OPCBIAS [V2] [subset_layer] in_layer [ref_layer]
[IGNORE jog_length]
[MINBIASLENGTH value [ [min_length_frag_corner] |
  [min_length_convex_corner min_length_concave_corner] ] |
  [ANGLED [angle_value [ [angle_min_length_frag_corner] |
    [angle_min_length_convex_corner angle_min_length_concave_corner] ]]] ]
[CLOSEREDGE [V1 | V2] [CLOSERSYMMETRIC]
[GRID grid_value]
[IMPLICITBIAS]
[OPTION ACUTE_ANGLE_ABUT [NO | YES]]
[OPTION IMPLICIT_METRIC [NO | YES]]
[OPTION FAST_CLASSIFY [NO | YES]]
[OPTION SPIKE_CLEANUP [NO | YES]]
[OPTION CAREFUL_SKEW [NO | YES]]
[OPTION CORNER_FILL [NO | YES]]
[OPTION PROJECTING_ONLY [NO | YES]]
[OPTION CAREFUL_MEASURE [NO | YES]]
{rule ...}
```

Where a **rule** is defined using either SPACE or SPACELAYER as follows:

- **SPACE constraint** [*metric*] [**WIDTH** *constraint* [*metric*]]  
[**WIDTH2** *constraint* [*metric*]] **MOVE** *value* [**LENGTH1** *constraint*]  
[**LENGTH2** *constraint*]
- **SPACELAYER** *layer MOVE value*

Product license: Calibre OPCpro or Calibre TDopc. Refer to the [Calibre Administrator's Guide](#) for complete licensing information.

### Summary

Performs both variable isolated/dense line and via biasing. Opcbias oversizes or undersizes objects by moving individual edges predefined amounts. If moving the edge causes discontinuity at the corners of abutting edges, Opcbias fills in the gaps to result in a smooth corner. Opcbias is used in conjunction with table-driven OPC. Unlike most SVRF operations, the order of parameters in the Opcbias operation is important and must be followed as shown.

---

#### Note



For a complete discussion of OPCbias use and table-driven OPC, refer to the [Calibre Rule-Based OPC User's Manual](#).

---

## Parameters

- **V2**  
Optional keyword that controls which hierarchical processing engine is executed for OPCbias: V1 or V2. V2 provides better scalability and uses the same streamlined hierarchical processing shared by Calibre nmOPC and Calibre OPCverify. Slight differences in outputs between V1 and V2 may also be observed. V2 must be enabled to use IGNORE, OPTION FAST\_CLASSIFY, and OPTION SPIKE\_CLEANUP.
- **subset\_layer**  
Optional input edge layer. This layer contains edges that require correction. It should be used when you need to apply OPC to only a subset of the original layer's edges. The *subset\_layer* must be derived from the input polygon layer.
- **in\_layer**  
Required input polygon layer that is subject to OPC. Typically this is a poly/gate or via/contact layer. When a *subset\_layer* is specified, only those edges that coincide with edges on the *subset\_layer* are corrected.
- **ref\_layer**  
An optional reference polygon layer used for spacing measurement. This layer contains all shapes with respect to which Calibre measures spacing. If this layer is specified, spacing is measured between the edge being corrected (on the *in\_layer*) and shapes on *ref\_layer*. If *ref\_layer* is not specified, spacing is measured between the edge and other polygons on the *in\_layer*.
- **IGNORE jog\_length**  
Optional keyword used to protect edges from biasing. If the length of an edge is less than or equal to *jog\_length*, then the edge will not get biased and it will not affect the biasing of other edges. If *jog\_length* is not specified, then all edges are eligible for measurement. This keyword is only available when using the V2 option.

---

### Caution



To use the IGNORE keyword, you must use the V2 processing mode. This is different than the V2 in CLOSEREDGE V2.

---

- **MINBIASLENGTH value [ [min\_length\_frag\_corner] | [min\_length\_convex\_corner min\_length\_concave\_corner] ] [ANGLED [angle\_value [ [angle\_min\_length\_frag\_corner] | [angle\_min\_length\_convex\_corner angle\_min\_length\_concave\_corner] ] ] ] ]**

Optional keyword specifying the minimum length for edges after biasing. You can also set the minimum length of fragments at corners (concave, convex, or all) and angled edges (concave, convex, or all).

The first argument is positive floating-point value specifying the minimum acceptable length for horizontal and vertical edges after biasing (including those edges that receive a bias of zero). There is no default value for *value*. If not specified, all edges are biased. When

possible, Calibre either widens edges shorter than *value* or merges them with their neighbors before biasing.

If two arguments are specified, the second value (*min\_length\_frag\_corner*) specifies the minimum length of fragments at corners.

If three arguments are specified, the second value (*min\_length\_convex\_corner*) specifies the minimum length of fragments at convex corners. The third argument (*min\_length\_concave\_corner*) specifies the minimum length of fragments at concave corners.

The secondary keyword ANGLED and the following positive floating-point argument (*angle\_value*) specifies a separate minimum acceptable length for angled edges. If not specified, *angle\_value* defaults to *value*. When ANGLED is specified, MINBIASLENGTH applies only to orthogonal edges.

If two arguments are specified after the ANGLED keyword, the second value (*angle\_min\_length\_frag\_corner*) specifies the minimum length of fragments at corners for angled edges.

If three arguments are specified after the ANGLED keyword, the second value (*angle\_min\_length\_convex\_corner*) specifies the minimum length of fragments at convex corners for angled edges. The third argument (*angle\_min\_length\_concave\_corner*) specifies the minimum length of fragments at concave corners at angled edges.

For a complete description of cleanup of short edges, refer to the [Calibre Rule-Based OPC User's Manual](#).

- **CLOSEREDGE [V1 | V2]**

An optional keyword that enforces a closer-edge-wins policy to determine which bias has a higher priority. The edge will receive treatment based on its priority and the lengths of neighboring edges.

If an edge is shorter than MINBIASLENGTH, the operation uses the closer edge wins condition and widens that edge into the fragment whose bias is closest to that short fragment. For a complete description of treating short edges between two long edges, refer to the [Calibre Rule-Based OPC User's Manual](#).

Specifying V1 or V2 selects either the original (V1) closer edge behavior or the short edge smoothing algorithm (V2). The V2 smoothing algorithm has better handling for flat versus hierarchical designs, rotational situations, and asymmetric cases.

CLOSEREDGE V2 (which can be specified as just “CLOSEREDGE”) is the default behavior for OPCbias, and is performed if the CLOSEREDGE V1 or CLOSERSYMMETRIC keywords are not specified.

- **CLOSERSYMMETRIC**

An optional keyword used to force symmetric widening when an edge is shorter than MINBIASLENGTH, the fragments to either side are longer than MINBIASLENGTH, and the short fragment has the higher priority.

**Caution**

To use the CLOSERSYMMETRIC keyword, you must use the default (V1) processing mode. This is different than the V1 in CLOSEREDGE V1.

---

Symmetric widening involves extending the short fragment by an equal amount in each direction, which shortens the adjacent fragments by equal amounts. This is the default behavior when the adjacent fragments have the same bias. This keyword instructs the operation to use this behavior even if the fragments on either side of the short fragment have different biases. To ensure proper biasing, use CLOSERSYMMETRIC only with reference layers that are a superset of the target layer.

- **GRID *grid\_value***  
Optional keyword and integer value that defines a reference grid for use when biasing 45-degree edges. When used, this parameter instructs OPCbias to adjust the bias of 45-degree edges so that they are on the specified grid. The *grid\_value* should be the same as the grid of the [Snap](#) operation.
- **IMPLICITBIAS**  
An optional keyword used to give edges with no rules specified an implicit bias of zero. IMPLICITBIAS is off by default.
- **OPTION ACUTE\_ANGLE\_ABUT [NO | YES]**  
An optional keyword used to apply bias to abutting acute angled edges. Abutting acute angled edges normally do not receive any bias. Setting this option to YES enables bias to be applied to those edges. This option is off by default.
- **OPTION IMPLICIT\_METRIC [NO | YES]**  
An optional keyword that uses the OPPOSITE\_EXTENDED metric for spacing rules that are created implicitly if the neighboring interval also uses the OPPOSITE\_EXTENDED metric. By default, all implicitly generated space rules use the OPPOSITE metric.
- **OPTION FAST\_CLASSIFY [NO | YES]**  
Optional keyword that, if enabled, uses an alternate edge classification algorithm to improve run time for non-hierarchical runs (specifically useful for section-based processing) or for large rule tables (hundreds of lines of rules). This does not change the output of OPCBIAS. This option is only available when using the [V2](#) option and is not compatible with rules using [LENGTH1](#), [LENGTH2](#), or [WIDTH2](#). This option is off by default.

**Caution**

To use OPTION FAST\_CLASSIFY, you must use the V2 processing mode. This is different than the V2 in CLOSEREDGE V2.

---

- **OPTION SPIKE\_CLEANUP [NO | YES]**

Optional keyword used to repair skewed edges on flat runs. This keyword is only available when using the V2 option. For more information on skewed edges, see “[Snapping for 45 Degree Angles](#)” in the *Calibre Rule-Based OPC User’s Manual*.

**Caution**



To use the OPTION SPIKE\_CLEANUP, you must use the V2 processing mode. This is different than the V2 in CLOSEREDGE V2.

- **OPTION CAREFUL\_SKEW [NO | YES]**

Optional keyword used to implement an improved algorithm that adds extra polygons to fill in gaps created by the boolean scanlines. For more information on skewed edges, see “[Snapping for 45 Degree Angles](#)” in the *Calibre Rule-Based OPC User’s Manual*.

- **OPTION CORNER\_FILL [NO | YES]**

Optional keyword used to control corner filling in OPCbias. For a figure illustrating the use of the CORNER\_FILL keyword, see [OPTION CORNER\\_FILL](#) in the *Calibre Rule-Based OPC User’s Manual*.

- **OPTION PROJECTING\_ONLY [NO | YES]**

Optional keyword used with OPPOSITE EXTENDED to classify an edge with a projected edge instead of a closer edge that is not projected. This option is off by default.

For a figure illustrating the use of the PROJECTING\_ONLY keyword, see [OPTION PROJECTING\\_ONLY](#) in the *Calibre Rule-Based OPC User’s Manual*.

- **OPTION CAREFUL\_MEASURE [NO | YES]**

Optional keyword used to reduce the difference between hierarchical and flat runs, especially if large values of OPPOSITE EXTENDED are used. This option can increase runtime and is off by default.

- ***rule***

A required rule, defining conditions and the movement to apply to edges that meet the conditions. You must specify at least one ***rule***; multiple rules are allowed. The order of ***rule*** parameters is important and must appear as shown in the syntax section. Rules are written using these parameters:

**SPACE *constraint*** — Required keyword and interval constraint (for example,  $> a < b$ ) in floating-point user units. An edge’s length must meet the SPACE condition in order for the ***rule*** to apply to an edge. The *constraint* may not be a strict equality (as in  $= a$ ). An unbounded interval is interpreted as the bounded complement of the unbounded interval. For example,  $> a$  is interpreted as everything not in the  $\leq a$  interval.

**SPACELAYER *layer*** — Required keyword and edge layer. Normally, when enforcing MINBIASLENGTH a closer-edge-wins principle is used. Since SPACELAYER rules have no information pertaining to space, the priority of the bias will come from the order that the rules are listed and an earlier-rule-wins principle will be used.

*metric* — Optional specification for how OPCbias evaluates a SPACE or WIDTH constraint. The OPCbias operation checks to see whether or not edges satisfy a SPACE or WIDTH constraint by calculating measurement regions and checking for edges that lie within these regions.

- For the SPACE condition, it creates measurement regions extending outside the figure. The portion of an edge that lies within an adjacent figure's measurement region is said to meet the condition.
- For the WIDTH condition, it creates measurement regions extending inside the figure. The portion of an edge that lies within the measurement region of an edge on the same figure is said to meet the condition.

WIDTH *constraint* — Optional keyword and interval constraint in floating-point user units that specifies the width a polygon feature must have in order for the *rule* to apply to an edge on that polygon feature. The interval constraint behavior is the same as for the SPACE constraint.

WIDTH2 *constraint* — Optional keyword and constraint that controls the selection of edges for biasing based on the widths of opposing polygon features. The interval constraint behavior is the same as for the SPACE constraint. If *constraint* is not specified, biasing is not impacted by the length of opposing edges.

MOVE *value* — Required keyword and floating-point number in user units that specifies the distance to move biased edges. A negative *value* causes biasing toward the interior of polygons. A positive *value* causes outward biasing.

LENGTH1 *constraint* — Optional keyword and constraint that controls selection of edges for biasing based upon their length. The constraint is in floating-point user units. Edges that satisfy the constraint are candidates for biasing. Edges that do not meet this condition are not considered in SPACE or WIDTH measurements. Strict equalities (for example,  $= a$ ) may not be used for the *constraint*.

LENGTH2 *constraint* — Optional keyword and constraint that controls selection of edges for biasing based upon the lengths of opposing edges used in a SPACE measurement. More precisely, if edge A projects upon edge B in such a way that the SPACE measurement can be taken between A and B, and edge A satisfies the LENGTH2 condition, then edge B is subject to biasing. LENGTH2 cannot be used with an unbounded SPACE constraint (as in  $> a$ ).

## Description

Biases *in\_layer* or *subset\_layer* edges based upon geometric relationships defined by the SPACE, WIDTH, WIDTH2, LENGTH1, and LENGTH2 conditions of a *rule*. The *rule* specifications set up table-driven OPC corrections. In order to ensure proper correction, each *rule* must be mutually exclusive. That is, every rule must differ from every other *rule* in at least one constraint.

## Opcbias Rule Syntax

You define basic edge conditions in terms of two properties of an input polygon's edge:

- The distance from the nearest objects. (SPACE parameter)
- The width of the polygon directly interior to that edge. (WIDTH parameter)

The syntax for a simple Opcbias **rule** is as follows:

**SPACE range WIDTH range MOVE distance**

Because each cell in an edge bias table represents a space, width, and associated movement, you create rules that account for every condition in your OPC table. Note that the conditions for SPACE and WIDTH in an Opcbias rule are ranges of distances and widths as opposed to discrete values:

```
SPACE <=0.150           WIDTH <=0.150           MOVE 0
SPACE > 0.200 <= 0.210 WIDTH > 0.150 <= 0.180 MOVE 0.5
```

The simple rules shown above contain only explicit table information. In most cases, your rules must include process constraints not stated explicitly in the table. To accurately translate the process into **rule** format, you must create rules that are more complex, using measurement metrics and length constraints.

Here are additional considerations for **rule** conditions:

**SPACE condition** — Edges are evaluated relative to edges on adjacent polygons either on the *in\_layer* or *ref\_layer*. For each adjacent polygon edge where a spacing measurement can be taken, Calibre calculates a measurement region based upon the *metric*. The portion of an edge that lies within an adjacent edge's measurement region is said to meet the SPACE *constraint*. In other words, the SPACE condition defines allowed distances to the nearest feature of interest on *in\_layer* or *ref\_layer*. Edges that meet the SPACE condition then undergo biasing if they satisfy the optional WIDTH and LENGTH conditions. If these optional conditions are not specified, then only the SPACE condition is checked.

**WIDTH condition** — Width is measured between opposing edges on the same polygon on the *in\_layer* or *subset\_layer*. For each opposing edge, a measurement region is calculated based upon the *metric*. The portion of an edge that lies within an opposing edge's measurement region is said to meet the WIDTH *condition*. If an edge meets the WIDTH condition, it can undergo biasing.

**WIDTH2 condition** — Controls the selection of edges for biasing based on the widths of opposing polygon features. The interval constraint behavior is the same as for the SPACE condition. If *condition* is not specified, biasing is not impacted by the length of opposing edges. More precisely, if edge A projects upon edge B in such a way that the SPACE measurement can be taken between A and B, edge B is subject to biasing only if:

- Edge B satisfies the WIDTH condition, if one is specified.
- Edge A satisfies the WIDTH2 condition, if one is specified.

**LENGTH1 condition** — Edges that satisfy this condition are subsequently tested against the SPACE and WIDTH conditions. For example, in the case of gates, you can use this condition to ensure that undesired edges on the boundary of gate and field poly are not subject to OPC biasing. To do this, use a *constraint* such as `> channel_length`, where `channel_length` is the gate width. In this way, the undesired poly edges are filtered out.

LENGTH1 places a constraint on the length of the edges subject to OPC for a particular *rule*. Only those edges that satisfy this condition can be corrected. MINBIASLENGTH defines the minimum acceptable length for an edge after the initial edge classification (including those edges that receive a bias of zero). MINBIASLENGTH applies to all rules processed by the operation, whereas the LENGTH1 constraint applies only to the *rule* for which it is supplied.

**LENGTH2 condition** — Edges that satisfy this condition are subject to OPC biasing. For example, you can use this condition in situations where you do not want narrow line-end edges to be used for the SPACE measurements. To do this, specify a constraint such as `narrow_line_end_length`, where `narrow_line_end_length` is the length of the line-end edge of the narrow polygon.

## Matching Rules to Tables

The SPACE, WIDTH, LENGTH1, and LENGTH2 conditions must be written in terms of a consistent partitioning of spaces and lengths:

- SPACE constraints partition external physical space relative to an edge (the distance between objects).
- WIDTH constraints partition internal space (width) relative to an edge.
- LENGTH1 constraints organize edges relative to length.
- LENGTH2 constraints organize opposing edges relative to length.

Within any set of rules, the partitioning must be consistent. To understand this concept, refer to [Table 4-33](#). The table itself is valid for edges of a specific length. The rows partition external physical space (SPACE condition). The columns partition internal space (WIDTH condition). Cell values in the table represent the biasing (MOVE parameter) to apply to any object that

meets the constraints for that row and that column. All rules written to represent this table define treatment based on the same partitioning of space.

**Table 4-33. Two-dimensional Rules Table**

Line Width		<=0.1	>0.1<=0.2	>0.2<=0.3	>0.3<=0.4	>0.4<=0.5	>0.5
Spacing	0.0	0	0	0	0	0	0
	>0<=0.1	0	0	0	0	-0.10	-0.15
	>0.1<=0.2	0	0	0	-0.05	-0.10	-0.10
	>0.2<=0.3	0	0	0	-0.05	-0.05	-0.10
	>0.3<=0.4	0	0.05	0	0	-0.05	-0.05
	>0.4	0	0.10	0.05	0.05	0	0

For a literal translation of the rules table into Opcbias rules, you would write one *rule* for each cell in the table. However, Opcbias allows you to write a single *rule* with constraints spanning multiple partitions, which is analogous to merging adjacent cells in a table. For example, in [Table 4-34](#), the four highlighted cells contain the same bias value. The single *rule*:

```
SPACE > 0.3 WIDTH > 0.2 <= 0.4 MOVE 0.05
```

represents all four cells.

**Table 4-34. Translating Rules Tables**

Line Width		<=0.1	>0.1<=0.2	>0.2<=0.3	>0.3<=0.4	>0.4<=0.5	>0.5
Spacing	0.0	0	0	0	0	0	0
	>0<=0.1	0	0	0	0	-0.10	-0.15
	>0.1<=0.2	0	0	0	-0.05	-0.10	-0.10
	>0.2<=0.3	0	0	0	-0.05	-0.05	-0.10
	>0.3<=0.4	0	.05	0.05	0.05	-0.05	-0.05
	>0.4	0	0.10	0.05	0.05	0	0

Opcbias does not allow you to split a partition one way for one *rule* and another way for another *rule*. For example, in the next two rules, the WIDTH partitions are split differently and cannot be used within the same Opcbias operation:

```
SPACE > 0 <= 0.1 OPPOSITE EXTENDED 0.1 WIDTH > 0.3 < 0.4 MOVE 0
SPACE > 0.1 <= 0.2 OPPOSITE EXTENDED 0 WIDTH > 0.3 <= 0.4 MOVE -0.05
```

Because the *metric* you use with either the SPACE or the WIDTH constraint plays a part in the partitioning, you must also make sure that the *metric* you use is consistent for the entire partition. For example, if you use one extension value for a particular WIDTH or SPACE

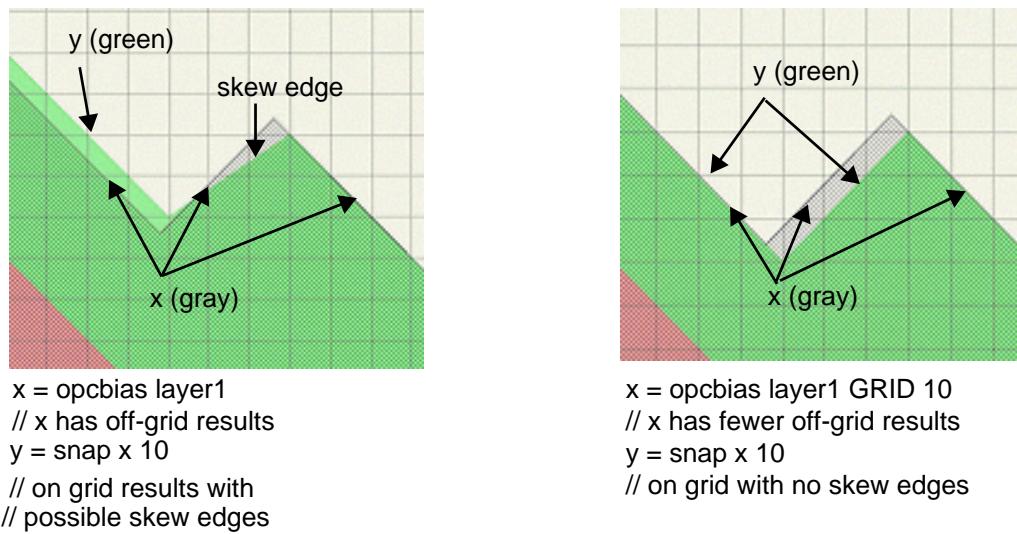
constraint, you must use the same extension value for all other occurrences of that constraint. The next two rules violate this rule and cannot be used within the same Opcbias operation.

```
SPACE > 0    <= 0.1 WIDTH > 0.3 <= 0.4 OPPOSITE EXTENDED 0.1 MOVE 0
SPACE > 0.1 <= 0.2 WIDTH > 0.3 <= 0.4 OPPOSITE EXTENDED 0.2 MOVE -0.05
```

### Using GRID

If you use the [Snap](#) operation to place Opcbias output on grid, you could see results where biased edges have their endpoints snapped on grid, but the edges are skew. Using the GRID option snaps the Opcbias output for most edges to the same grid the Snap operation uses. There will still be some edges from an Opcbias GRID operation that are off grid due to internal edge breaking; however, any such off-grid edges are then placed on grid by Snap. Figure 4-216 shows an example.

**Figure 4-216. Opcbias GRID Option**



### Examples

The following is a rule check that covers column 3 of [Table 4-34](#) (assuming poly as the input layer):

```
column_3 { opcbias poly
    SPACE <= 0.3           WIDTH > 0.1 <= 0.2   MOVE 0
    //covers first 4 cells
    SPACE > 0.3 <= 0.4   WIDTH > 0.1 <= 0.2   MOVE 0.05 //cell5
    SPACE > 0.4           WIDTH > 0.1 <= 0.2   MOVE 0.10 //cell6
}
```

If you want to ensure that no edges are generated that are less than 0.05 in the above rule check, you would specify MINBIASLENGTH 0.05.

# Opclineend

Layer operation

**OPCLINEEND** [V2] *in\_layer* [*via\_layer1* BY *value1* [*via\_layer2* BY *value2*]]  
 [RLAYER *reference\_layer*]  
 [CORNERFILL *fill1* [*fill2*] [ITERATIONS *iter*]]  
 {*rule\_set* ...}

Where a *rule\_set* is defined as follows:

```
{WIDTH constraint [HEIGHT constraint] END value_list
SERIF extension_list depth_list
[WIDTH constraint [HEIGHT constraint] {SPACE constraint [metric]}]
END value_list SERIF extension_list depth_list
...}
```

Product license: Calibre OPCpro or Calibre TDopc. Refer to the *Calibre Administrator's Guide* for complete licensing information.

## Summary

Provides variable line-end biasing by allowing you to define line-end extensions, serifs, and spacing conditions. Unlike most SVRF operations, the order of the parameters for Opclineend is important and must be followed as shown.

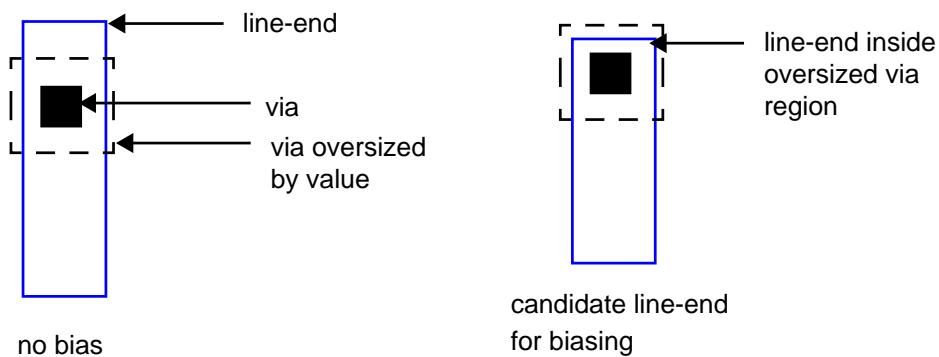
### Note



For a complete discussion of Opclineend usage and table-driven OPC, refer to the *Calibre Rule-Based OPC User's Manual*.

## Parameters

- V2  
Optional keyword that enables the V2 hierarchical processing engine for OPCLINEEND. V2 enables an improved hierarchical promotion scheme that produces the same results, and, in ambiguous cases, ensures the same results as the flat results. V1 is the default mode of operation.
- *in\_layer*  
Required input polygon layer that is subject to OPC.
- *via\_layer1* BY *value1* [*via\_layer2* BY *value2*]  
Optional polygon layer, keyword, and floating-point distance used for calculating which line-ends receive OPC based on existence of vias near the line-ends. The *value1*, specified in user units, defines how close a line-end must be to a via for them to be considered near each other. Calibre uses the *value1* to calculate an oversize region for the via. Line-ends that fall within the oversize region are candidates for receiving OPC. [Figure 4-217](#) provides a graphical description of this process. You can specify up to two via layers.

**Figure 4-217. Via Oversizing**

- **RLAYER *reference\_layer***

Optional keyword and polygon layer that indicates the layer on which all line-ends are found. All spacing checks are performed with respect to the *reference\_layer*, if you specify it. The *reference\_layer* should be a superset of the *in\_layer*; however, this dependency is not checked at compilation time. If you do not specify RLAYER, the *in\_layer* is used to check spacing.

- **CORNERFILL *fill1* [*fill2*] [ITERATIONS *iter*]**

Optional control over whether to apply corrections to notched line-ends. When specified, the operation fills the notch prior to performing line-end extension. CORNERFILL applies only when at least one *via\_layer* is present.

The arguments *fill1* and *fill2* define the maximum notch size allowed in a line-end receiving correction.

- *fill1* — maximum dimension in direction 1. If only *fill1* is specified, the operation fills notches with dimensions up to *fill1* x *fill1*.
- *fill2* — maximum dimension in direction 2. If both values are specified, the operation fills notches with dimensions up to *fill1* x *fill2*.

Use the keyword ITERATIONS to control the number of iterations performed in the event that there is a stair-step at the line-end. The default for ITERATIONS is 1.

- ***rule\_set***

Required rule set, defining conditions and the line-end extensions to apply to line-ends that meet the condition. You must specify at least one *rule\_set*. Multiple rules in a *rule\_set* are allowed. The first rule in a *rule\_set* is called the default rule. Additional rules, if specified, are called correction rules. When multiple rules are specified in a *rule\_set*, each rule's WIDTH/HEIGHT condition pair must be mutually exclusive with respect to all other rules in the set; constraint ranges cannot overlap. These are the parameters of a *rule\_set*:

- **WIDTH *constraint***

Required keyword and width condition that a line-end must meet in order for the rule to apply to that line-end. The WIDTH condition defines the width of the polygon at the

line-end, which can also be described as the length of the edge at the line-end. The ***constraint*** is specified in user units as an interval of non-negative real numbers. The ***constraint*** must have an upper bound (that is,  $< x$  or  $\leq x$  must appear in the constraint). If you specify multiple WIDTH intervals in the same rule check, the WIDTH constraint intervals must be mutually exclusive.

- **HEIGHT *constraint***

Optional keyword and height condition that a line-end must meet in order for the rule to apply to that line-end. The HEIGHT defines the height of the polygon at the line-end, which can also be described in terms of the lengths of the edges that share endpoints with the line-end edge. The ***constraint*** is specified in user units as an interval of non-negative real numbers.

A single HEIGHT condition defines the condition for both adjacent edges to the line-end. [Table 4-35](#) defines how the condition relates to each of the edges. If the HEIGHT condition is not specified, the default is  $> 0$ .

**Table 4-35. Height Conditions and Edges**

HEIGHT <i>constraint</i>	Example	Implementation	Edge 1	Edge 2
Open interval, bounded above	$\leq a$	At least one edge must be less than upper bound	$\leq a$	$> 0$
Closed interval	$> a \leq b$	Both edges must be greater than the lower bound and at least one edge must be less than upper bound	$> a \leq b$	$> a$
Open interval, bounded below	$> a$	Both edges must be greater than the lower bound	$> a$	$> a$

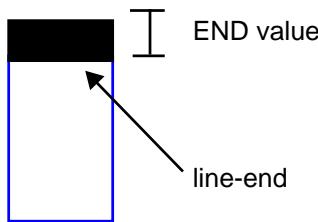
- **END *value\_list***

Required keyword and distance the line-end is to be moved to create a line-end extension. The ***value\_list*** elements are specified in non-negative floating-point user units.

For default rules, END specifies the initial movement of a line-end, which is the maximum movement in the ***rule\_set***. Only one value may appear in a default rule.

For correction rules, END specifies the movement used to avoid a spacing violation that can be created by the default rule. Correction rule values are used instead of default rules where spacing violations occur. The END specification for a correction rule can have up to three values in the list. These values are discussed in the “Rule Set Details” section beginning on [page 1250](#).

The END condition is required for all rules. [Figure 4-218](#) shows an illustration.

**Figure 4-218. Line-End Extension**

- **SERIF extension\_list depth\_list**

Required keyword and non-negative floating-point numbers in user units that define serifs. The *extension\_list* and *depth\_list* are applied to add a serif to the **END** extension.

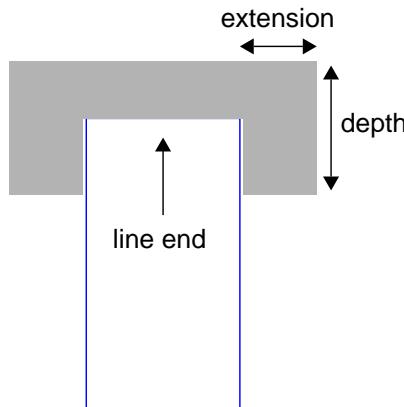
The serif extension is applied along the direction of the line-end. The depth is applied perpendicular to the extension. As with the **END** condition, the **SERIF** condition specifies the initial edge movements for default rules and specifies the edge movements used to avoid a spacing violation for correction rules. Correction rule values are used instead of default rules where spacing violations occur.

The *extension* and *depth* in a default rule are the maximum serif extension and depth applied, and can contain *only* one pair of values.

Correction rules can have up to three values in the *extension\_list* and three values in the *depth\_list*. These values are discussed in the “Rule Set Details” section. Correction rule extensions can only be less than the default rule *extension*. The default *depth* can be increased in correction rules to keep the serif a constant overall depth in the event the line-end is shortened by the correction rule.

If either **SERIF** extension or depth are zero, only the line-end extension is created (for a default rule) from the **END** condition (if non-zero), as in [Figure 4-218](#). For a correction rule, the **SERIF** extension and depth replace the corresponding values of the default rule.

In the case where both the serif extension and depth are non-zero, the system creates a hammerhead by generating a serif shape in addition to the line-end extension. Thus, *depth* defines the length of the hammerhead in the direction opposite of the line-end extension, and *extension* defines the serif width. [Figure 4-219](#) shows an example.

**Figure 4-219. Serif Dimensions**

- **SPACE constraint [metric]**

Required keyword and constraint, with an optional metric that defines a spacing violation condition in a correction rule (note that correction rules are optional). Not used in a default rule. When a spacing violation exists, Calibre replaces the default line-end extension with the associated correction rule extension to correct for that violation.

You must specify the *constraint* in user units as an interval of non-negative floating-point numbers. Unbounded space conditions (for example,  $> 0.7$ ) are interpreted as the complement of the bounded counterparts (that is, everything not in the interval  $\leq 0.7$ ). Spacing intervals in a *rule\_set* that apply to line-ends having the same WIDTH must be mutually exclusive.

The SPACE condition supports an optional measurement *metric*. The default is OPPOSITE SYMMETRIC. You may specify OPPOSITE EXTENDED *value*. Metrics are discussed in detail in the *Calibre Verification User's Manual* and the *Calibre Rule-Based OPC User's Manual*.

## Description

Performs variable line-end extension, from simple end extension to hammerhead creation. With this operation, you can instruct the Calibre engine to create line-end extensions based on line-end length and height in such a way to ensure that the generated extensions do not cause spacing violations. Opclineend is typically used for table-driven OPC.

In general, a *rule\_set* contains a default rule and one or more correction rules. The default rule defines the maximum OPC. Correction rules define the OPC treatment required for line-ends that would introduce spacing violations if they were corrected using the default rule. The following example shows a *rule\_set* comprised of the default rule plus two correction rules.

```
poly_opc {opclineend poly
width < 0.3 height > 0.2           end 0.02      serif 0.02 0.1
//default
width < 0.3 height > 0.2 space <= 0.3    end 0.005     serif 0      0.115
// first correction
```

```

width < 0.3 height > 0.2 space >= 0.3 <= 0.35 end 0 0.001 serif 0.01 0.11
// second correction
}

```

All three rules in the example apply to poly line-ends of width less than 0.3 and height greater than 0.2. In the default rule, the line-end extension is 0.02, the serif extension is 0.02, and the serif depth is 0.1.

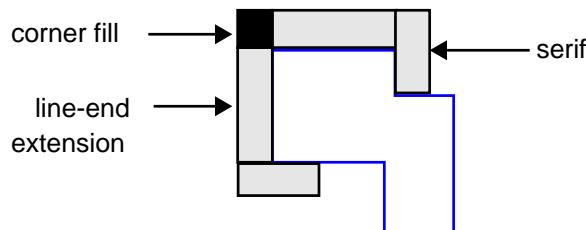
The first correction rule is applied when the default rule introduces a spacing error, such that the spacing between adjusted edges is less than or equal to 0.3. For edges that meet the spacing condition, the line-ends are moved 0.005, and no serifs would be formed because the serif extension is zero.

The second correction rule is applied when the default rule introduces a spacing error of the interval  $> 0.3 \leq 0.35$ . You can see what the line-end and serif values are in the example for this case.

Notice the intervals of the SPACE condition are mutually exclusive. This must be true in any *rule\_set*.

In the case where line-end edges are abutting, no serif is generated at the abutting corners. Instead, corner gap filling operation is conducted for the line-end extension shapes at such corners. See [Figure 4-220](#).

**Figure 4-220. Corner Filling**



## Rule Set Details

You must specify at least one *rule\_set* as part of the Opclineend command.

Rule sets are written in two parts:

- **Default rules** — The default rule defines a condition, based on line-end width and (optionally) height, the extension to apply to all line-ends that meet that condition, and the serif dimensions. The default rule is required. Notice that the default rule does not contain a SPACE condition.
- **Correction rules** — Zero or more correction rules define spacing violation conditions and extensions designed to avoid these violations. Calibre measures spacing violations *after* applying the default rule. When it encounters a violation, it replaces the default line-end and serif extensions, and serif depth, with the extensions and depths defined by the correction rules. Correction rules adhere to the same general syntax as default rules, except that they contain the additional SPACE violation condition and support lists of values for **END** and **SERIF** conditions.

The **END** condition can contain up to three values, and the **SERIF** condition can contain up to three values for extension and three values for depth. These account for up to three different spacing violations a default rule can introduce. These are discussed in detail in the following paragraphs.

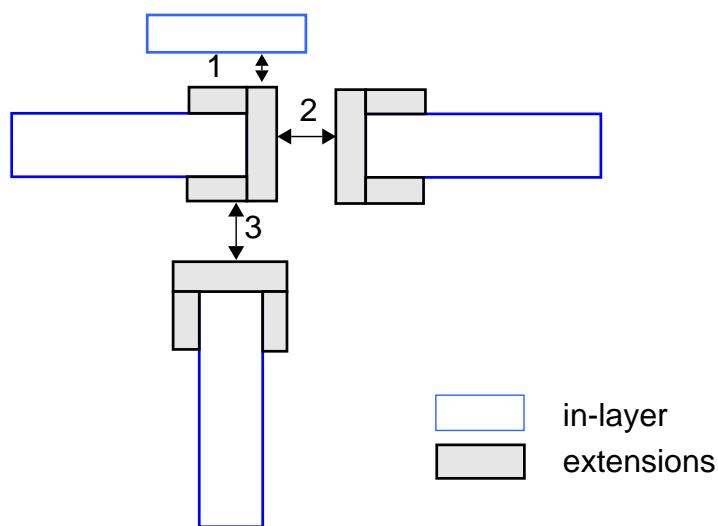
Correction rules must use the same **WIDTH** and **HEIGHT** conditions specified in the default rule for the rule set.

Calibre uses the **SPACE** condition in correction rules to correct for three cases of spacing violations. These are:

1. A non-adjusted opposing edge to an edge that has been extended. (An edge that has not received OPC is involved in the violation.)
2. An opposing edge is extended in the same way as an extended edge. (Line-end-to-line-end or serif-to-serif violation.)
3. An opposing edge is extended in a different way than an extended edge. (Line-end-to-serif violation.)

[Figure 4-221](#) shows these cases.

**Figure 4-221. Types of SPACE Violations**



The **END** condition in a correction rule can have up to three values specified.

- If only one value appears, it applies to all three spacing violations in [Figure 4-220](#).
- If two values appear, the first value applies to case 1, and the second value applies to cases 2 and 3.
- If three values appear, each applies correspondingly to cases 1 through 3.

The **SERIF** condition in a correction rule can have up to three values for extension, followed by up to three values for depth. These values correspond to [Figure 4-220](#) in the same way as the **END** values do.

For example, consider this correction rule:

```
WIDTH < 0.3 SPACE < 0.05 END 0 0.05 0.05 SERIF 0 0.05 0.05 0.1 0.15 0.15
```

The **END** specification has three extension correction values. The **SERIF** specification has three extension correction values, followed by three depth correction values. These values correspond to the three cases in [Figure 4-220](#), respectively.

The number of values you supply for **END** extensions and **SERIF** extensions need not match. However, the number of **SERIF** extension values and **SERIF** depth values must match to avoid ambiguity.

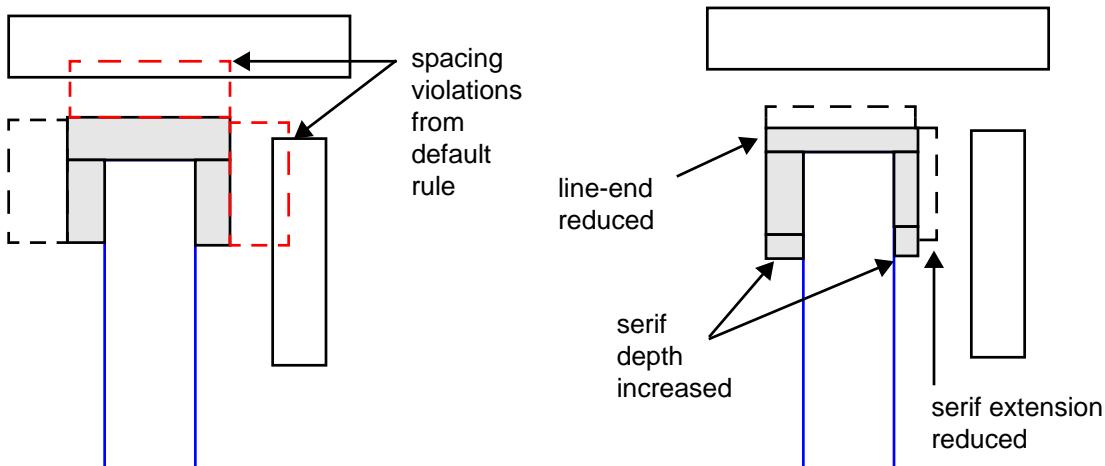
## Rule Set Guidelines

Opclineend requires one rule for each **WIDTH** constraint in your rules table. Each *rule\_set* contains one default rule. Unless your table contains only one spacing constraint, the *rule\_set* must also contain one correction rule for each spacing constraint in the rules table.

**Writing Default Rules** — The default rule defines the maximum OPC. The syntax for a default rule is described previously. **END** and **SERIF** extension values in the default rule define the maximum movements defined for a given width. Depending on whether the total serif length is constant or variable, the serif depth will either be:

- constant serif length minus the maximum line-end movement
- your choice of serif length table values minus the maximum line-end movement

**Writing Correction Rules** — Correction rules define the actual OPC treatment required for line-ends that would introduce spacing violations if they were corrected using the default rule. After applying the default rule, if Opclineend encounters a spacing violation, it replaces the default corrections with the corrections defined in the correction rule. [Figure 4-222](#) illustrates this process. Notice that the movement values in the correction rule are specified with respect to the original line-end, not the line-end after any correction.

**Figure 4-222. Applying Correction Rules**

The SPACE constraint in a correction rule represents both types of spacing violations: line-end violations and side-edge violations. Spacing intervals for correction rules covering the same WIDTH parameter in a *rule\_set* must be mutually exclusive.

Because Opclineend checks for violations after applying the default rule, you must adjust the spacing values accordingly in the tables before you can use them as correction rules.

## Examples

### Example 1

In this example, the default extension for line-ends and serifs is 0.15, the serif depth is a constant 0.3, and the minimum spacing to enforce is 0.20. The drawn geometry grid is 0.05.

```
x = opclineend poly
width <= 0.2                                end 0.15      serif 0.15 0.3 // 1
width <= 0.2 space <= 0.05                  end 0          serif 0    0.3 // 2
width <= 0.2 space > 0.05 <= 0.10           end 0.05 0.10 serif 0.05 0.10 0.3 0.3
                                                // 3
width <= 0.2 space > 0.10 <= 0.15           end 0.10      serif 0.10 0.3 // 4
```

Line 1 is the default rule. Line 2 covers a spacing violation of 0.05 units or less (recall the grid is 0.05). No line-ends or serifs are formed by this rule.

Line 3 covers a spacing violation between 0.05 and 0.10 units. For case 1 violations (see [Figure 4-220](#)) of this rule, the default extension plus the maximum spacing violation equals 0.25, which is 0.05 more than the minimum spacing you want to enforce. So, a line-end or serif can move up to 0.05 units and not cause a violation for case 1. For case 2 and 3 violations, both edges moved 0.15 units to create the maximum spacing violation of 0.10. So,  $2(0.15) + 0.10 = 0.40$  is the original distance between the edges. Because both edges involved in a spacing violation move in cases 2 and 3, they can each move 0.10 units and not violate your minimum spacing of 0.20. The serif depth is a constant 0.3 in all cases.

Line 4 covers a spacing violation of up to 0.15 units. For case 1 violations of this rule, the default extension plus the maximum spacing violation equals 0.30. This is 0.10 more than the minimum spacing you want to enforce. So, a line-end or serif can move up to 0.10 units and not

cause a violation in this case. For case 2 and 3 violations, both edges moved 0.15 units and there is a maximum of 0.15 units separating them. Hence,  $2(0.15) + 0.15 = 0.45$  is the original distance between the edges. In theory, both edges could move up to 0.125 units and not violate the minimum spacing of 0.20. However, our grid is to the nearest 0.05, so we must choose 0.10 for the correction. Therefore, the 0.10 correction applies in all cases for this rule.

### Example 2

Multiple rule sets.

```

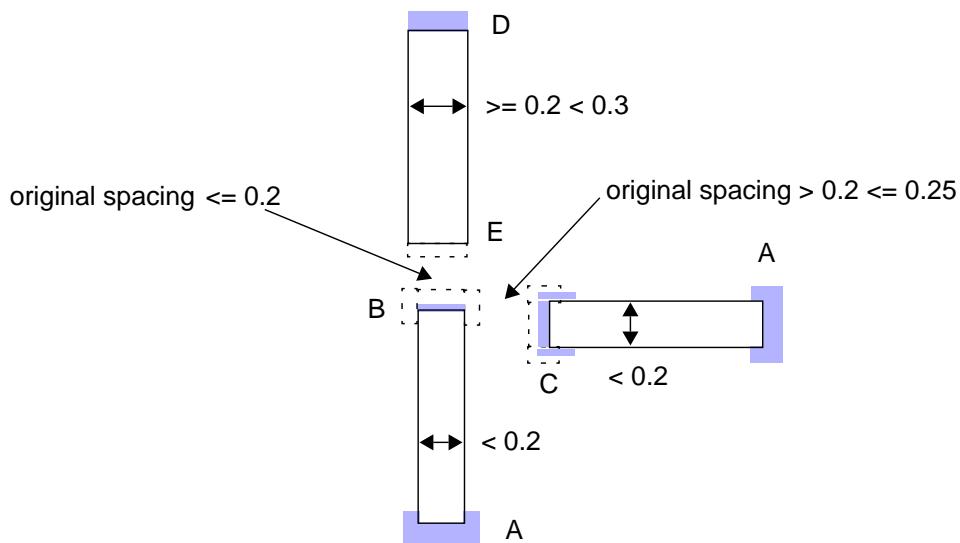
poly_serif { opclineend poly
    // 1st rule set. WIDTH and HEIGHT constraints match.
    // SERIF values take into account changes in END values.

    // rule A
    WIDTH < 0.2 HEIGHT > 0.1                      END 0.02  SERIF 0.02 0.1
    // rule B
    WIDTH < 0.2 HEIGHT > 0.1 SPACE <= 0.2          END 0.005 SERIF 0      0.115
    // rule C
    WIDTH < 0.2 HEIGHT > 0.1 SPACE > 0.2 <= 0.25 END 0.01   SERIF 0.01 0.11

    // 2nd rule set. WIDTH interval is mutually exclusive with first rule set.

    // rule D
    WIDTH >= 0.2 < 0.3 HEIGHT > 0.1                  END 0.02  SERIF 0      0
    // rule E
    WIDTH >= 0.2 < 0.3 HEIGHT > 0.1 SPACE <= 0.2 END 0      SERIF 0      0
}

```



# Opcsbars

Layer operation

**OPCSBAR** *in\_layer* [*ref\_layer*]  
 {[SBLAYER *sblayer* [CLEANSBLAYER]}  
 [OFFSETLAYER *offsetlayer*]  
 [V1]  
 [OPTION VERIFICATION\_MODE [NO | YES]]  
 [MINEDGELENGTH *value*]  
 [MINSBSpace *value*] [MINSBOFFSET [*condition*] *value* [OPPOSITE]]  
 [LINEENDOFFSET *value*] [LINEENDSPACE *value*] [LINEENDTOLONGSPACE *value*]  
 [MINSBWIDTH *value*] [MAXSBWIDTH *value*]  
 [MINSBLENGTH *value*] [MAXSBLENGTH *value*]  
 [WITHWIDTH] [NEGATIVE] [ANGLED]  
 [INTERSECTION {N | L [*u\_bottom\_length*] | T | P}] [STRICTCENTER]  
 [HORIZONTAL] [VERTICAL] [ANGLEEDGE]  
 [OPPOSITEEXTENDED *value*]  
 [OPEN]  
 [MINJOGWIDTH *min\_overlap*]  
 [JOG FILL [*dist width*]]]  
 [PRIORITY BY LAYER *layer1* [*layerN ...*]]  
 [VIALAYER]  
 [CENTERMERGE *value width*] [LINEENDMERGE *value*]  
 [EXTENSION *value* [BEFORECLEANUP]]  
 [COLINEARIZE *length width*]  
 [SMOOTH *value*]  
 [OPTION CAREFUL\_CLEANUP [0 | 1]]  
 [OPTION FAST\_MRC [NO | YES]]  
 [OPTION PRIORITIZE\_SPACE [NO | YES]]  
 [NOCLEANUP [*priority*]]  
 [PRIORITYCENTER]  
*rule* [*rule ...*]

Where a **rule** has one of two mutually exclusive forms:

- **SPACE** *constraint* [WIDTH *constraint*]  
 {{SBWIDTH *value* {SBOFFSET *value* | SBPITCH *value*} | CENTER [*center\_offset*] | CENTERPITCH [*center\_offset*]}}...}
- **SPACELAYER** *layer* {{SBWIDTH *value* SBOFFSET *value*}...}

Product license: Calibre OPCsbar. Refer to the [Calibre Administrator's Guide](#) for complete licensing information.

## Summary

Creates scattering bars for optical process correction. Use of this statement is discussed in the [Calibre OPCsbar User's Manual](#).

## Parameters

- ***in\_layer***

Required input layer that is subject to scattering bar correction. When used alone (without a *ref\_layer*) ***in\_layer*** must be an original or derived polygon layer. When used with a *ref\_layer*, ***in\_layer*** can be either an original or derived polygon layer, or a derived edge layer.

- ***ref\_layer***

Optional reference layer used for spacing measurement. This layer contains all shapes with respect to which Calibre measures spacing. The *ref\_layer* must be a superset of the ***in\_layer***; that is, it must contain all shapes on the ***in\_layer***. If *ref\_layer* is specified, spacing is measured between an edge on the input layer (***in\_layer***) and edges on this layer. If *ref\_layer* is not specified, spacing is measured between edges on the ***in\_layer*** alone. You cannot specify *ref\_layer* with SPACELAYER.

- **SBLAYER *sblayer* [CLEANSBLAYER]**

An optional polygon layer containing pre-existing scattering bars to be factored into calculations when generating additional scattering bars. The OPCbar operation uses this layer two ways:

During cleanup:

- Assigns scattering bars on *sblayer* the highest priority.
- Factors this geometry into all spacing checks.
- If you include the optional CLEANSBLAYER, the operation performs a full cleanup on the *sblayer* with respect to the ***in\_layer*** and *ref\_layer* and other shapes on the *sblayer*. This cleanup is performed after all operation-created scattering bars have been cleaned up. It evaluates the shapes on *sblayer* with respect to all cleanup options, including colinearize, lineendmerge, extension, jogfill and correcting violations to the space and offset rules.

As output:

- Combines geometry on *sblayer* with newly-created scattering bars and outputs a single merged layer form.
- **OFFSETLAYER *offsetlayer***

An optional polygon layer that defines areas where scattering bars cannot be created.

The actual area in which scattering bars are prohibited is greater than the polygons on the *offsetlayer*, because OPCbar also checks for MINSBOFFSET and LINEENDOFFSET violations between scattering bars and polygons on the offset layer. If any violations occur, they are cleaned up. This is in addition to the removal of violations between scattering bars and edges from ***in\_layer***.

- **V1**

An optional keyword for OPCsbar that controls whether V2 or V1 mode is executed. V2 is the default mode of operation as it provides better scalability and improved performance over V1. V2 supports creation of negative SRAF as center scattering bars and HORIZONTAL, VERTICAL, and ANGLEEDGE style controls. If NEGATIVE or CENTER are specified, the keyword WIDTH is required. Slight differences in outputs between V1 and V2 may be observed.

---

**Note**

 From 2007.2 to 2008.2, positive scattering bars were processed in the default V2 mode of operation but negative scattering bars invoked the V1 engine. Starting with the 2008.2 release, both positive and negative scattering bars are processed in the V2 mode of operation.

---

V1 is available as an optional environment variable for users who prefer to maintain output characteristics shown in releases prior to 2007.2. The V1 keyword can be used directly in the OPCsbar portion of the rule file or enabled with the following environment variable in the command line before invoking Calibre:

```
setenv OPCSBAR_ENABLE_V1 TRUE
```

- **OPTION VERIFICATION\_MODE [NO | YES]**

An optional keyword used to find deleted scattering bars of an executed OPCsbar operation on the sblayer, and places these scattering bars on a separate layer. Take the following outline, for example:

```
my_deleted_sbars = OPCSBAR input
MRC RULES
SBAR recipe
OPTION VERIFICATION_MODE YES
SBLAYER my_sbars
```

In this outline, scattering bars that can be placed on the my\_sbars layer (and still satisfy the MRC rules) will be placed on the my\_deleted\_sbars layer.

- **MINEDGELENGTH *value***

Optional minimum length of an edge in order for it to be considered a valid boundary edge for which scattering bars are generated. This length applies to edges after they have been classified based upon the SPACE conditions defined in the rules. The default is 0.

- **MINSBSPACE *value***

Optional minimum spacing requirement between adjacent scattering bars created by this operation. The *value* specifies the minimum floating-point spacing. If not specified, MINSBSPACE defaults to the value of MINSOFFSET.

- **MINSOFFSET [*condition*] *value* [OPPOSITE]**

Optional minimum offset requirement for all scattering bars created by this operation. The optional *condition* is a range of floating-point values that specify the widths of the scattering

bars for which the rule will be applied. The *value* specifies the minimum floating-point distance allowed between any scattering bar and original polygon edges. If not specified, MINSBOFFSET defaults to the smallest SBOFFSET value in any **rule**. This offset is not enforced for perpendicular edges. MINSBOFFSET is enforced identically for positive and negative scattering bars.

When optional conditions are specified, different offset values can be applied to scattering bars of different widths. Multiple conditions are only available in the V2 (default) mode of operation.

The OPPOSITE keyword causes the operation to use the OPPOSITE measurement metric to enforce offset spacing. The default is to use the Euclidean distance. Refer to the “[metric](#)” section for more information.

- LINEENDOFFSET *value*

Optional minimum offset requirement for all line-end edges of scattering bars created by this operation. The *value* specifies the minimum floating-point distance allowed between any scattering bar line-end and original polygon edges. The *value* must be  $\geq 0$ , with 0 implying a line-end may touch an original feature. A scattering bar line-end is defined as  $\geq \text{MINSBWIDTH}$  and  $\leq \text{MAXSBWIDTH}$ . If not specified, LINEENDOFFSET defaults to MINSBOFFSET.

- LINEENDSPACE *value*

Optional minimum spacing requirement between the line-end edge of one scattering bar created by this operation and the line-end edges of two scattering bars. The *value* specifies the minimum floating-point spacing. If not specified, LINEENDSPACE defaults to MINSBSPACE.

- LINEENDTOLONGSPACE *value*

Minimum spacing requirement between the line-end edge of one scattering bar created by this operation and the long edge of another scattering bar. If not specified, LINEENDTOLONGSPACE defaults to LINEENDSPACE.

- MINSBWIDTH *value*

Optional minimum width requirement for all scattering bars created by this operation. The *value* specifies the minimum floating-point width. If not specified, MINSBWIDTH defaults to the smallest SBWIDTH value in any rule -1 database unit.

- MAXSBWIDTH *value*

Optional maximum width requirement for all scattering bars created by this operation. The *value* specifies the maximum floating-point width. If not specified, MAXSBWIDTH defaults to the largest SBWIDTH value in any rule +2 database units.

- MINSBLENGTH *value*

Optional minimum length requirement for all scattering bars created by this operation. The floating-point *value* applies to the total length of a scattering bar, not necessarily to each segment. If not specified, MINSBLENGTH defaults to MAXSBWIDTH +2 database units.

In addition, MINSBLENGTH serves as the minimum length for valid boundary edges. Calibre does not generate scattering bars adjacent to any orthogonal (with respect to the database axes) edge smaller than MINSBLENGTH.

- **MAXSBLENGTH** *value*

Optional maximum length requirement for all scattering bars created by this operation. The *value* specifies the maximum floating-point length. By default, MAXSBLENGTH is off until *value* is specified.

- **WITHWIDTH**

An optional flag defining the method used to classify edges based on width.

- When WITHWIDTH is not specified, widths are classified using the Internal ... OPPOSITE operation.
- When WITHWIDTH is specified, widths are classified using the With Width operation.

- **ANGLED**

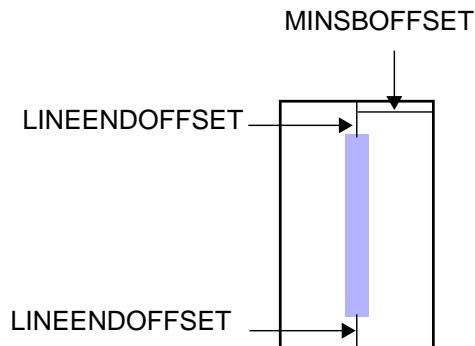
Optional keyword controlling generation of scattering bars adjacent to non-orthogonal (with respect to database axes) boundary edges. By default, Calibre only generates scattering bars adjacent to orthogonal edges. If the keyword ANGLED is supplied, Calibre also generates scattering bars adjacent to 45-degree edges.

Because Calibre builds scattering bars for angled edges out of small abutting or overlapping edges, these scattering bars are removed during cleanup unless JOG FILL is also specified.

- **NEGATIVE**

NEGATIVE is an optional flag that instructs the OPCsbar operation to create negative scattering bars. The default is to create positive scattering bars. This keyword controls all scattering bars created by the operation; you cannot create both negative and positive scattering bars in a single operation. This operation affects all scattering bars created by OPCsbar.

To create partial-cut negative scattering bars, use MINSBOFFSET and LINEENDOFFSET as shown in [Figure 4-223](#).

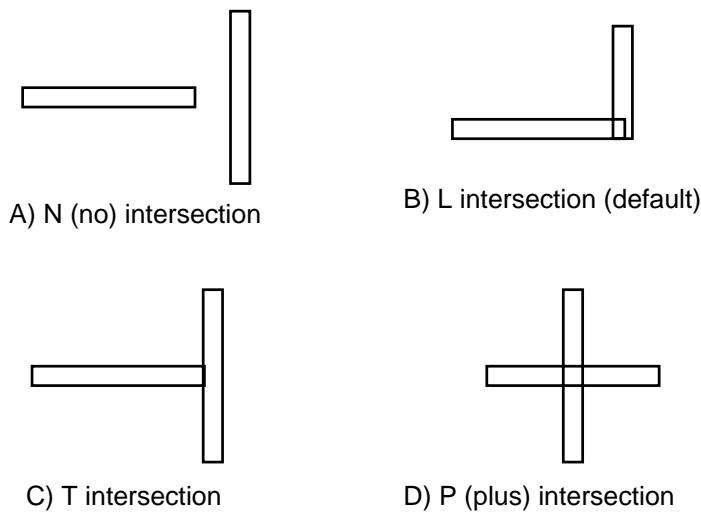
**Figure 4-223. Partial-Cut Negative Scattering Bars**

- INTERSECTION N | L [*u\_bottom\_length*] | T | P

An optional keyword set controlling the types of intersections allowed between scattering bars. The four valid intersection types are:

- N — no intersections allowed.
- L — L-shaped intersections allowed.
- T — T-shaped and L-shaped intersections allowed.
- P — + -shaped, T-shaped, and L-shaped intersections allowed.

The default is L, which corresponds to a joined corner. [Figure 4-224](#) illustrates the four different intersection types: N, L, T, and P.

**Figure 4-224. INTERSECTION Options**

The optional *u\_bottom\_length* parameter for the L keyword is a positive, floating-point number, interpreted as a minimum length in user units of the bottom of a U-shaped polygon. If two L-shaped polygons abut to form a U-shaped polygon, the distance between the

parallel bars of the polygon is measured. If that distance is less than *u\_bottom\_length*, the “bottom” of the U is removed.

When Calibre encounters two scattering bars intersecting in a way that is not allowed, it trims the overlaps (defined as less than or equal to MINSBLENGTH), and cuts away the intersection as needed to create an allowed intersection.

- **STRICTCENTER**

An optional argument that determines whether a scattering bar is generated between two edges, if the SPACE condition is met and only if one WIDTH condition is satisfied. If STRICTCENTER is not specified, a scattering bar is placed in the center between both edges. When STRICTCENTER is specified, a scattering bar is not generated.

**Note**

STRICTCENTER and OPPOSITEEXTENDED cannot be used together.

- **HORIZONTAL**

An optional flag that enables the application of the present rule only to those edges derived from horizontal features. Vertical or angled edges that are classified by this rule are ignored unless a similar rule exists specifying the VERTICAL or ANGLEEDGE style controls. This option is only available in the V2 (default) mode of operation.

- **VERTICAL**

An optional flag that enables the application of the present rule only to those edges derived from vertical features. Horizontal or angled edges that are classified by this rule are ignored unless a similar rule exists specifying the HORIZONTAL or ANGLEEDGE style controls. This option is only available in the V2 (default) mode of operation.

- **ANGLEEDGE**

An optional flag that enables the application of the present rule only to those edges derived from angled features. Vertical or horizontal edges that are classified by this rule are ignored unless a similar rule exists specifying the VERTICAL or HORIZONTAL style controls. This option is only available in the V2 (default) mode of operation.

- **OPPOSITEEXTENDED *value***

An optional flag that defines the method used to classify edges based on space. This operation affects all scattering bars created by OPCsbar. When OPPOSITEEXTENDED is specified, edges participating in space searches are extended by the specified *value*. This feature is only available in V2 mode.

There should be at least one SPACE classification when using OPPOSITEEXTENDED.

**Note**

OPPOSITEEXTENDED and STRICTCENTER cannot be used together.

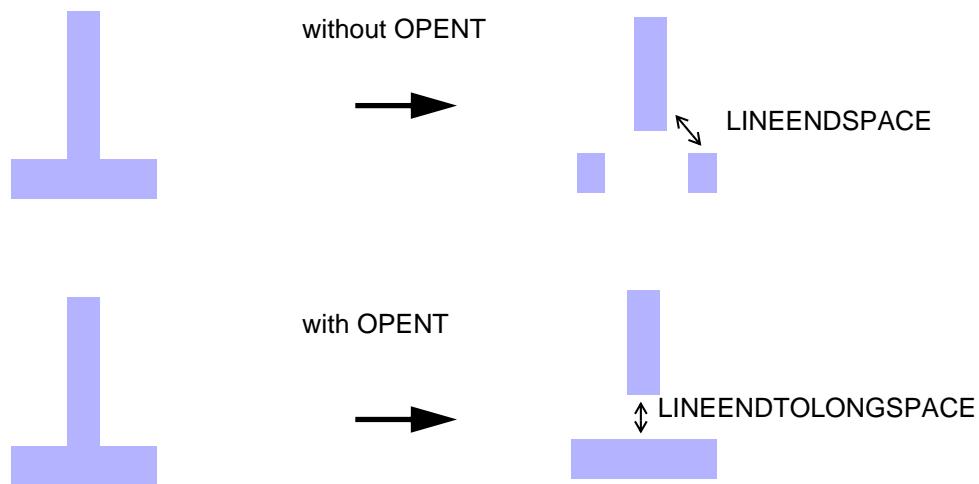
For the definition of the OPPOSITE EXTENDED metric, see the “Metrics” section in the *Calibre Verification User’s Manual*.

- **OPENT**

An optional keyword controlling how OPCsbar cleans up “T” intersections when no intersection is allowed.

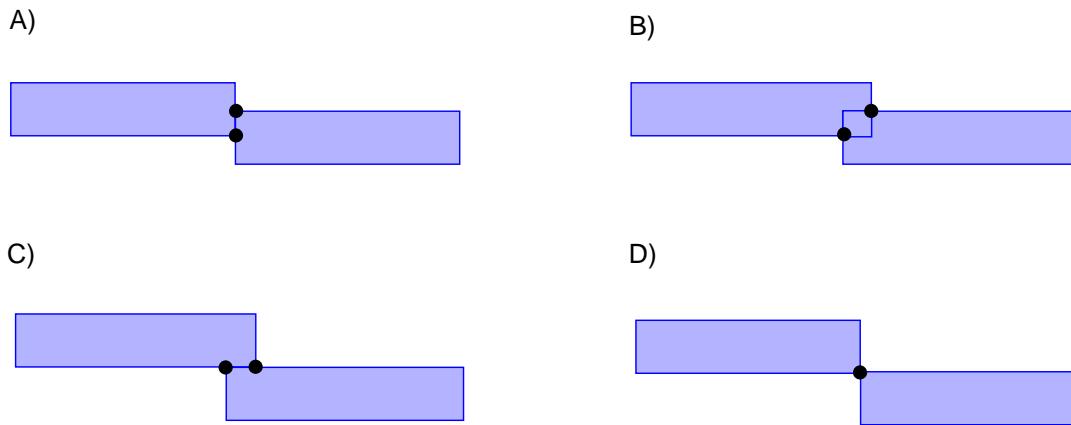
By default, the OPCsbar cleanup operation trims the center of the intersection away equally. If you specify OPENT, the OPCsbar cleanup operation trims away stems of Ts.

**Figure 4-225. INTERSECTION N Cleanup With and Without OPENT**



- **MINJOWIDTH *min\_overlap***

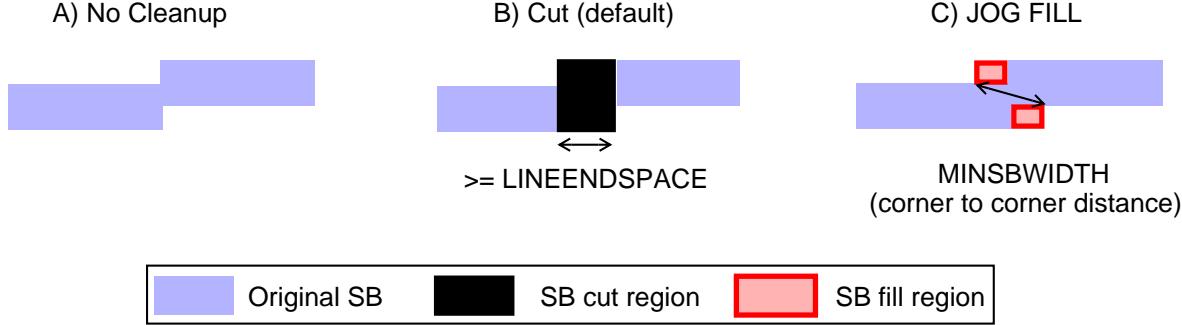
Defines the minimum overlap required for a jog to be legal. OPCsbar compares the size of the overlap between scattering bars, and if it is smaller than *min\_overlap*, the cleanup operation cleans up the jog. If it is larger, the jog is considered to be legal and the cleanup operation leaves it alone. The figure that follows shows four overlap configurations. In all cases except the singularity (configuration D) the overlap distance is measured as the distance between the two points shown.

**Figure 4-226. Measuring Jog Overlaps**

The *min\_overlap* must be a floating point number  $\leq \text{MAXSBWIDTH}$ . If not specified, the default is as follows:

- Without JOG FILL the default is 0.
- With JOG FILL the default is *MINSBWIDHT*.
- JOG FILL [*value [width]*]  
Optional keyword with optional, floating-point value in user units of length, and optional, floating-point width, controlling the method used to clean up overlapping scattering bars, touching scattering bars, singularities, or scattering bars within *value* distance of another.

By default, Calibre cuts the scattering bars, trimming them back from the initial contact to form a legal separation. Specifying JOG FILL allows Calibre to extend both scattering bars equally to ensure that the resulting overlap is equal to *MINSBWIDHT*.

**Figure 4-227. Styles of Jog Interaction**

If the optional *value* is specified, then a legal jog is formed between adjacent (not touching) scattering bars that lie within *value* distance of each other. The *value* must be greater than or equal to *MINSBLENGTH*. The application fills in both directions. When scattering bars are

configured such that there are multiple ways of adding jog fill, the application chooses the configuration arbitrarily.

If the optional *width* is specified, scattering bars of that width are created when resolving violations. The default is MINSBWIDTH.

Jog filling only occurs between scattering bars of equal priority. If there are multiple ways of choosing jog fill configuration, Calibre arbitrarily chooses the configuration.

Because Calibre creates scattering bars for angled edges by combining many small abutting or overlapping scattering bars, JOG FILL has significant impact on whether or not these special scattering bars remain after cleanup.

- **PRIORITIZE BY LAYER** *layer1 [layerN...]*

Optional keyword set and list of layers that defines a prioritization scheme to use when resolving intersections or spacing violations. By default, the system assigns the highest priority to the scattering bars that are closest to the input geometry edges, second highest priority to the next closest scattering bars, and so on. Prioritization rankings are assigned after the initial cleanup of scattering bars that do not meet the MINSBLENGTH constraint.

When PRIORITIZE BY LAYER is specified, Calibre prioritizes the edges based on interaction with additional input layers, called island layers. You must supply at least one island layer, which must contain polygon data.

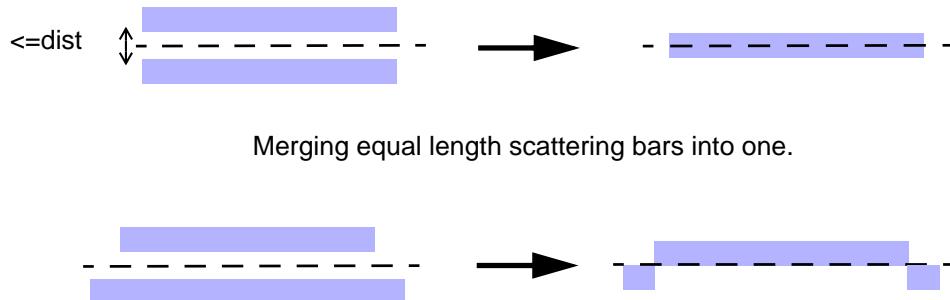
- **VIALAYER**

An optional keyword instructing the OPCsbar cleanup operation to use the VIALAYER cleanup method. With this cleanup method, the OPCsbar cleanup operations trims away some of the higher priority scattering bar when doing so makes it possible for both scattering bars to remain after cleanup.

- **CENTERMERGE** *value [width]*

Optional keyword, floating-point value in user units, and optional width that specifies scattering bars separated by a distance less than *value* should be merged on the centerline between the two bars. The two scattering bars to be merged must have the same priority, but not the same length. [Figure 4-228](#) shows examples of how CENTERMERGE merges scattering bars.

**Figure 4-228. CENTERMERGE**



If you specify *width*, this defines the width of scattering bars when resolving violations. The default for CENTERMERGE is MAXSBWIDTH.

- **LINEENDMERGE *value***

Optional keyword and floating-point value used to instruct Calibre to merge scattering bars when they meet the following constraints:

- The line-ends of the two scattering bars are lined up perfectly, so that no jogs are created by merging.
- The distance between the two scattering bars is less than or equal to *value*.

The default is to not merge.

- **EXTENSION *value* [BEFORECLEANUP]**

Optional keyword and non-zero floating-point value. A positive *value* instructs Calibre to extend line-ends of scattering bars by the specified amount. When *value* is negative, Calibre shortens valid ends by the specified amount. A line-end is  $\geq$  MINSBWIDTH and  $\geq$  MAXSBWIDTH. The original scattering bar has a greater priority than an extension during the cleanup phase by default.

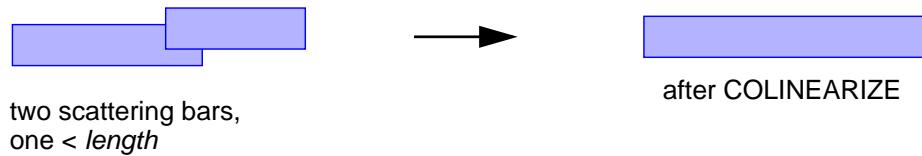
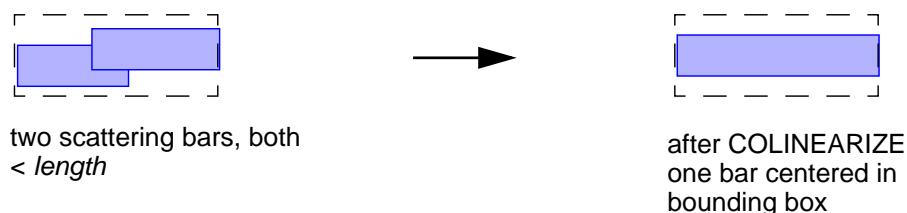
The BEFORECLEANUP keyword causes extensions to be generated when scattering bars are generated. This means the scattering bars and associated extensions have the same priority for the cleanup phase.

- **COLINEARIZE *length* [*width*]**

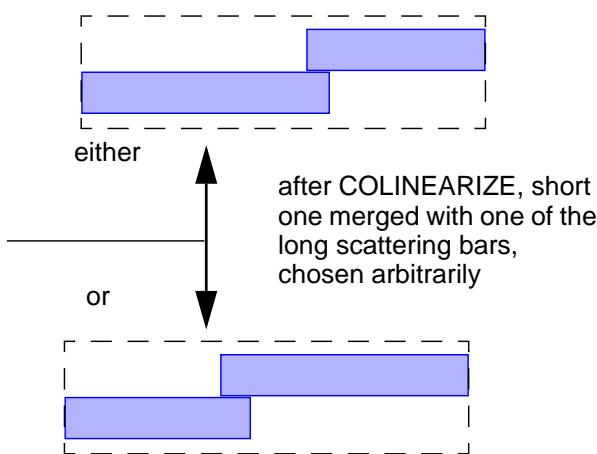
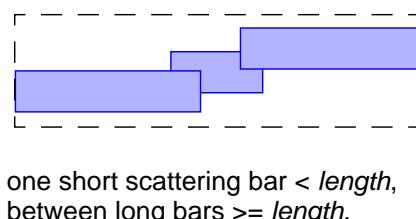
Optional keyword with positive, floating-point length in user units, and optional, floating-point width, that specifies to align and merge overlapping scattering bars that are offset from each other if at least one of the bars is less than *length* units long. In order for two scattering bars to participate, they must both have the same priority. Singularities do not constitute overlap. Merging by the COLINEARIZE option is performed before JOG FILL.

Merging is performed as follows:

- If only one scattering bar is shorter than *length*, it is aligned and merged with the bar that is equal to or longer than *length*.
- If both scattering bars are shorter than *length*, a bounding box is drawn around the extent of both bars. The bars are replaced by a single bar of MINSBWIDTH, centered within the bounding box.
- If a scattering bar shorter than *length* lies between two scattering bars greater than or equal to *length*, then the short bar is merged with an arbitrarily-chosen longer bar.

**Figure 4-229. COLINEARIZE****Case 1****Case 2****Case 3**

(If JOG FILL is Specified)



The optional *width* specifies that scattering bars of that width are created when resolving violations. The default is MINSBWIDTH. This is only used in the second case shown in [Figure 4-228](#). In the first case in the figure, the width of the longer scattering bar is used.

- **SMOOTH *value***

An optional keyword used to smooth out small features created by edge classification. If the length of an edge is less than *value*, the edge will be merged with its neighbor using

CLOSEREDGE. For layers such as contact, it is recommended that *value* is set to the original layer edge length.

- **OPTION CAREFUL\_CLEANUP [0 | 1]**

An optional keyword specifying the level that scattering bars will be recaptured. Scattering bars are recaptured by comparing the original raw scattering bars to the cleaned up scattering bars. The CAREFUL\_CLEANUP option is only available in the V2 (default) mode of operation.

This option can also be set using the OPCSBAR\_CAREFUL\_CLEANUP environment variable although the option takes precedence over the environment variable setting. To set this option using an environment variable, set OPCSBAR\_CAREFUL\_CLEANUP to 0 in a shell:

```
setenv OPCSBAR_CAREFUL_CLEANUP 0
```

- **OPTION FAST\_MRC [NO | YES]**

An optional keyword that, if enabled, skips the last pass of cleanup which cleans MRC residues in the design. To enable FAST\_MRC, OPTION CAREFUL\_CLEANUP must be set to 1.

This option can also be enabled using the OPCSBAR\_FAST\_MRC environment variable although the option takes precedence over the environment variable setting. To enable this option using an environment variable, set OPCSBAR\_FAST\_MRC to YES in a shell:

```
setenv OPCSBAR_FAST_MRC YES
```

- **OPTION PRIORITIZE\_SPACE [NO | YES]**

An optional keyword that, if enabled, prioritizes negative scattering bars based on the SPACE condition in a rule. Negative scattering bars generated from smaller SPACE condition have a higher priority than negative scattering bars generated from larger SPACE condition. Negative scattering bars that are generated with the same SPACE condition have the same priority regardless of any WIDTH conditions.

This option can also be enabled using the OPCSBAR\_PRIORITIZE\_SPACE environment variable although the option takes precedence over the environment variable setting. To enable this option using an environment variable, set OPCSBAR\_PRIORITIZE\_SPACE to 1 in a shell:

```
setenv OPCSBAR_PRIORITIZE_SPACE 1
```

- **NOCLEANUP [*priority*]**

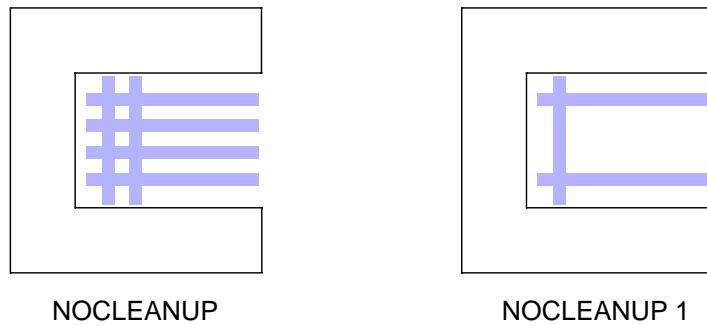
An optional keyword that instructs Calibre to stop processing after creating the raw scattering bars and performing extensions and line-end merging. When specified, Calibre does not enforce spacing or width rules and does not enforce allowed intersection rules.

When used without the optional *priority*, Calibre returns all raw scattering bars.

The optional *priority* is a number Calibre assigns at runtime to the scattering bars it creates. The highest priority scattering bars have the priority 1, while the lowest priority scattering

bars have the largest assigned priority number. When used with the optional *priority*, Calibre returns only the raw scattering bars with this priority number. If *priority* is greater than the highest priority number assigned at runtime, Calibre returns the raw scattering bars with the largest assigned priority number. [Figure 4-230](#) shows an example.

**Figure 4-230. Using NOCLEANUP**



- **PRIORITYCENTER**

By default, Calibre performs Prioritization by Order where the highest priority scattering bars are those closest to the valid edge. The next highest priority scattering bar are those that are the second closest to the valid edge and so on. **PRIORITYCENTER** is an optional keyword that gives the center scattering bars the highest priority during the cleanup process instead of the scattering bar closest to the valid edge. When using this option, OPCsbar maintains center scattering bars at the expense of non-center scattering bars.

- **rule**

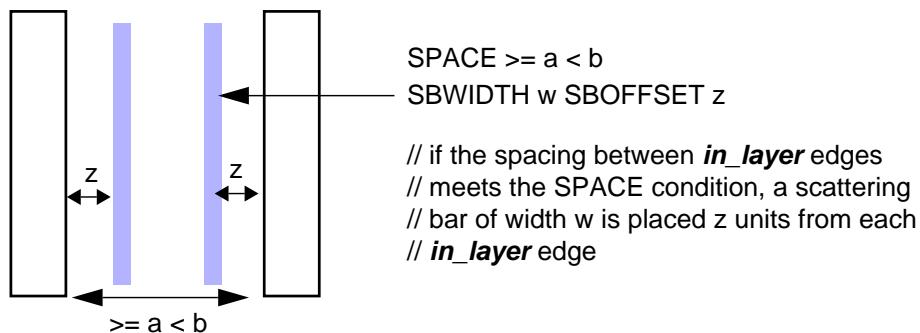
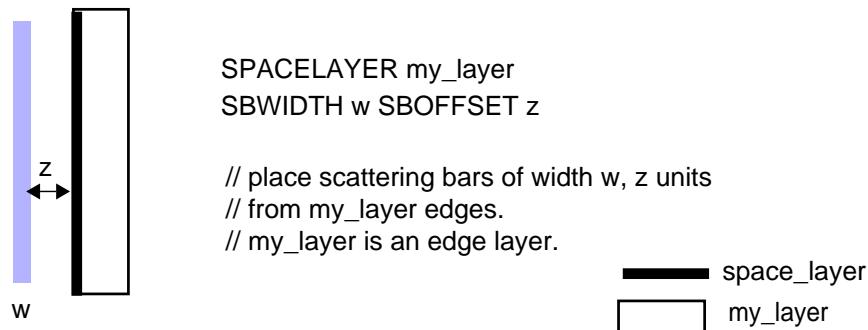
Required rule defining a space condition or space layer, and the width and placement of each scattering bar to be created according to the conditions of the **rule**. Rules correspond to scattering bar table entries. At least one rule is required. Multiple rules are allowed. Rules have two mutually-exclusive forms with parameters described in the section “[Rule Syntax](#).”

### Rule Syntax

A **rule** has one of two mutually exclusive forms, **SPACE** or **SPACELAYER**. That is, you cannot combine these forms within a single OPCsbar operation. Rules must all be **SPACE** or must all be **SPACELAYER**.

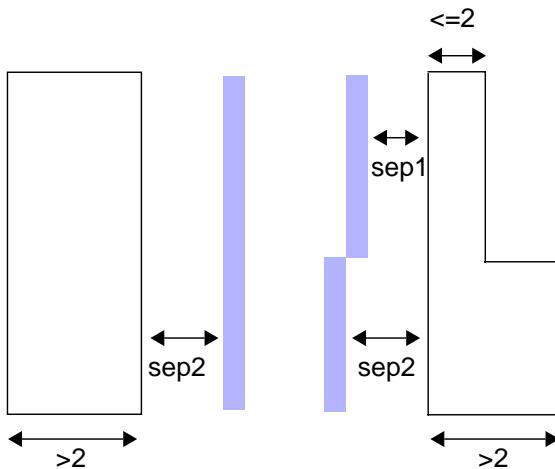
#### SPACE

**SPACE** *condition* [WIDTH *condition*] { {SBWIDTH *value* {SBOFFSET *value* | SBPITCH *value* | CENTER [*center\_offset*] | CENTERPITCH [*center\_offset*]} } ... }

**Figure 4-231. Typical SPACE Rule****SPACELAYER****SPACELAYER** *layer* {{SBWIDTH *value* SOFFSET *value* }...}**Figure 4-232. Typical SPACELAYER Rule****SPACE and SPACELAYER Parameters**

The parameters for the two rule forms are described here.

- **SPACE constraint [WIDTH constraint]** — Required keyword and constraint specifying the required space condition in order for the rule to apply. When you use the optional WIDTH constraint with the SPACE constraint, an edge must also meet this constraint in order for the rule to apply.

**Figure 4-233. WIDTH for Positive Scattering Bars**

```
SPACE <=space1 WIDTH <=2 SBWIDTH 0.08 SBOFFSET sep1
SPACE <=space1 WIDTH > 2 SBWIDTH 0.08 SBOFFSET sep2
```

You must specify each of the ***constraints*** in user units as an interval of non-negative floating-point numbers. Unbounded space conditions (for example,  $> 0.7$ ) are interpreted as the complement of the bounded counterparts (that is, everything not selected with the  $\leq 0.7$  constraint).

Space constraint ranges must be mutually exclusive; that is, the constraints in a rule cannot overlap. You cannot use the SPACE and SPACELAYER parameters in the same ***rule***.

- **SPACELAYER *layer*** — Required keyword and layer name, where *layer* is an original or derived edge layer. The layer is used to calculate the placement of the scattering bars. You cannot use this parameter with *ref\_layer*, SPACE, or CENTER parameters.
- **SBWIDTH *value*** — Required keyword and floating-point value that create one scattering bar of width *value* adjacent to the valid *in\_layer* edges (if used in a SPACE rule), or adjacent to SPACELAYER edges (if used in a SPACELAYER rule). The associated placement argument, SBOFFSET, SBPITCH, CENTER, or CENTERPITCH defines the placement of the scattering bar.

You can specify multiple SBWIDTH parameters for each SPACE or SPACELAYER rule. You must specify SBWIDTH for each scattering bar, even if all scattering bars are the same width.

- **SBOFFSET *value*** — A required argument with SPACELAYER or an optional argument with SPACE that defines the placement for scattering bars created using this rule. If used in a SPACE rule, the nearest edge of a scattering bar is placed the SBOFFSET value from any valid *in\_layer* edge. If used in a SPACELAYER rule, the nearest edge of a scattering bar is placed the SBOFFSET value from any edge on the SPACELAYER.

- For positive scattering bars, the distance is measured between the scattering bar and the exterior side of the nearest valid *in\_layer* edge or SPACELAYER edge. This is the default.
- For negative scattering bars, the distance is measured between the scattering bar and the interior side of the nearest valid *in\_layer* edge or SPACELAYER edge.

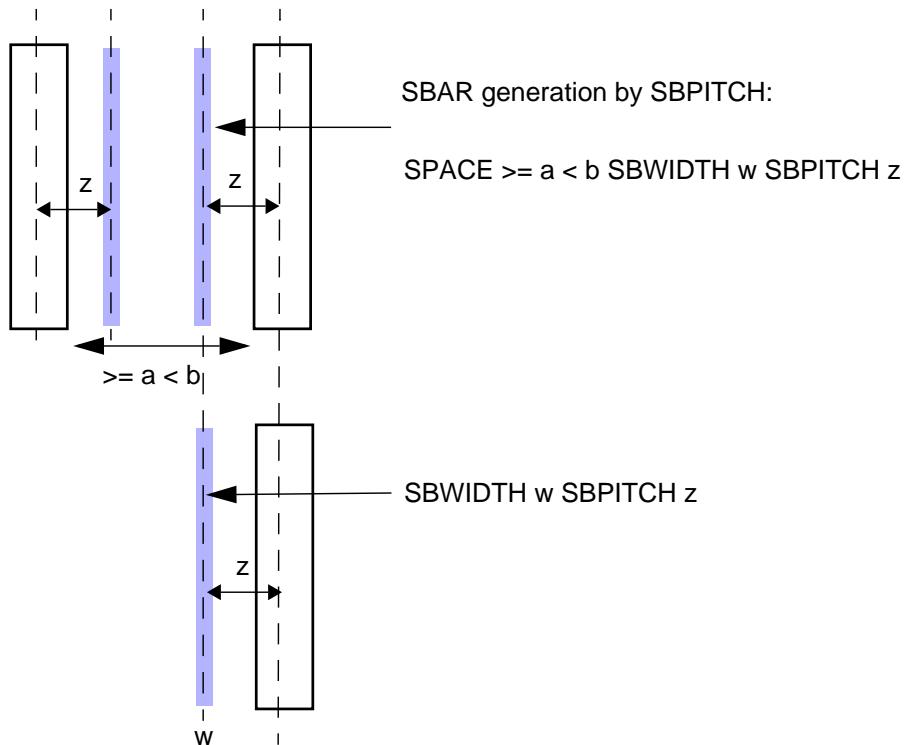
You must specify one placement argument (SBOFFSET, SBPITCH, CENTER, or CENTERPITCH) for each set of scattering bars created.

- **SBPITCH value** — Used only with the SPACE form, this option defines the placement for scattering bars created using this rule, expressed as the distance between the scattering bar center-line and the center of the feature at the nearest valid edge. SBPITCH requires a constrained WIDTH specification that must be satisfied in order for the rule to apply. Only positive scattering bars are supported with SBPITCH. This option is only available in the V2 (default) mode of operation.

You must specify one placement argument (SBOFFSET, SBPITCH, CENTER, or CENTERPITCH) for each set of scattering bars created.

This option is applied in addition to the existing style controls and edge selection rules. Depending on the rule and the data, each SBPITCH triggers the creation of either one scattering bar or two, as shown in [Figure 4-234](#).

**Figure 4-234. SBPITCH**



- **CENTER** [*center\_offset*] — Used only with the SPACE form, this option defines the placement for the scattering bar of width SBWIDTH to be an equal distance between the two, valid *in\_layer* edges that satisfy the SPACE condition.

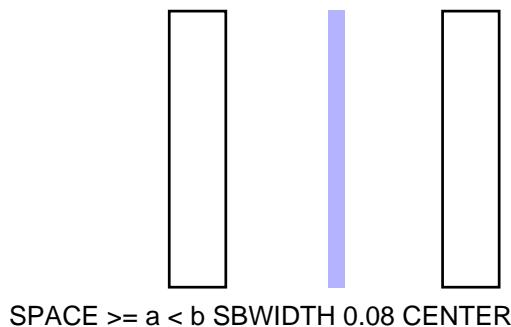
You must specify one placement argument (SBOFFSET, SBPITCH, CENTER, or CENTERPITCH) for each SBWIDTH. CENTER is used only with the SPACE parameter, not SPACELAYER.

- If you define a WIDTH constraint, the application only creates a centered scattering bar if *both edges* meet the criteria for the same rule. That is, same WIDTH category, same SPACE category.
- CENTER supports negative scattering bars if V2 and a WIDTH constraint are specified.

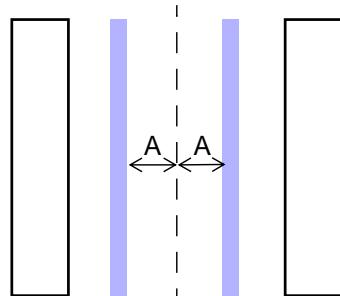
You can only specify one CENTER placement argument for each SPACE condition. Thus, if you have several SBWIDTHs for a SPACE condition, only one can use the CENTER placement argument.

When used with the optional *center\_offset*, this argument defines the placement for two scattering bars of width SBWIDTH, and placement relative to the centerline of the space between the two *in\_layer* edges. The *center\_offset* defines the distance from the centerline. [Figure 4-235](#) shows an illustration.

**Figure 4-235. CENTER**



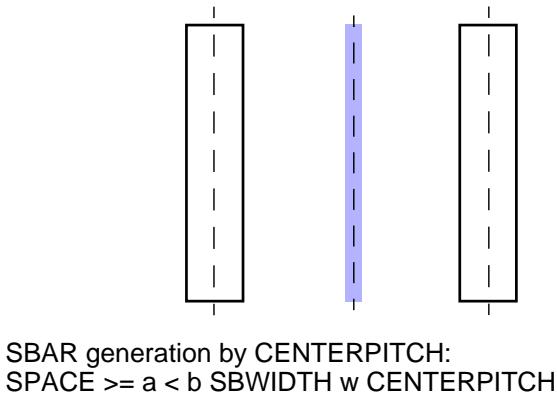
SPACE >= a < b SBWIDTH 0.08 CENTER



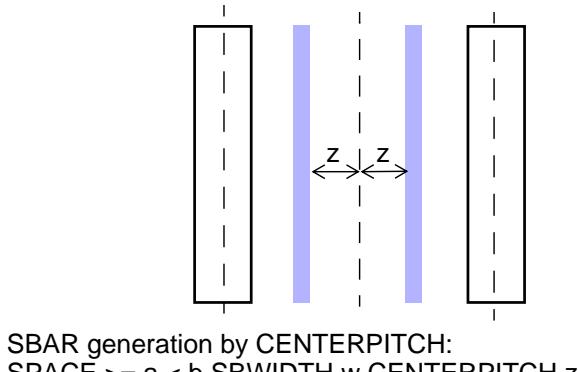
SPACE >= a < b SBWIDTH 0.08 CENTER A

- **CENTERPITCH** [*center\_offset*] — Used only with the SPACE form, this option defines the placement for the scattering bar of width SBWIDTH to be an equal distance between the two, valid *in\_layer* edges that satisfy the SPACE condition. Only positive scattering bars are supported with CENTERPITCH. This option is only available in the V2 (default) mode of operation. [Figure 4-236](#) illustrates the behavior of CENTERPITCH.

**Figure 4-236. CENTERPITCH**



SBAR generation by CENTERPITCH:  
SPACE  $\geq a < b$  SBWIDTH w CENTERPITCH



SBAR generation by CENTERPITCH:  
SPACE  $\geq a < b$  SBWIDTH w CENTERPITCH z

- You must specify one placement argument (SBOFFSET, SBPITCH, CENTER, or CENTERPITCH) for each SBWIDTH. CENTERPITCH is used only with the SPACE parameter, not SPACELAYER.
- You can only specify one CENTERPITCH placement argument for each SPACE condition. Thus, if you have several SBWIDTHs for a SPACE condition, only one can use the CENTERPITCH placement argument.
- If you define a WIDTH constraint, the application only creates a centered scattering bar if *both edges* meet the criteria for the same rule. That is, same WIDTH category, same SPACE category.

- CENTERPITCH does not support negative scattering bars.
- When used with the optional *center\_offset*, this argument defines the placement for two scattering bars of width SBWIDTH, and placement relative to the centerline of the space between the two *in\_layer* edges. The *center\_offset* defines the distance from the centerline.

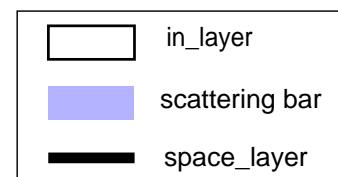
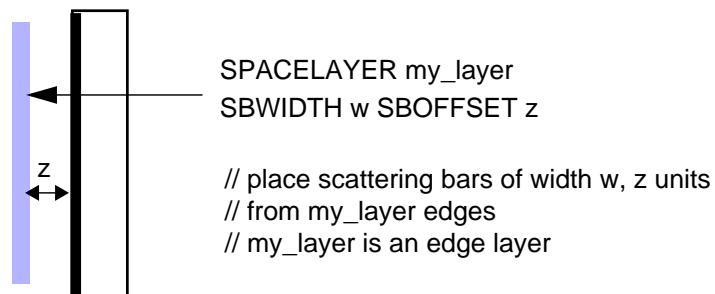
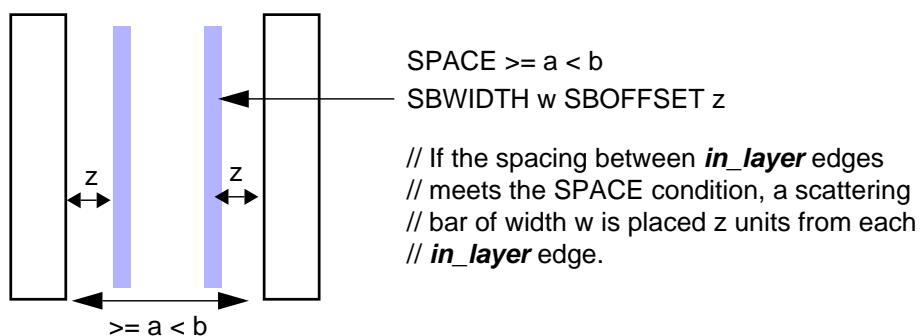
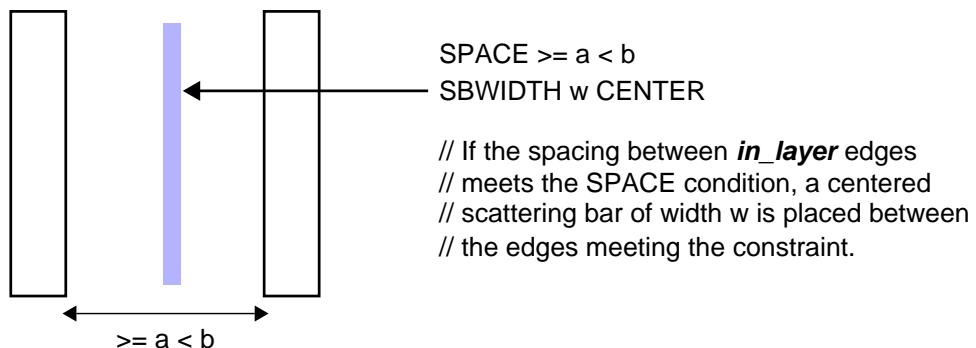
## Description

Automates scattering bar creation by generating a derived polygon layer consisting of scattering bars. OPCsbar performs all the following tasks:

- Identifies portions of the mask design that require scattering bars.
- Creates scattering bars according to user-defined rules.
- Checks for spacing and width violations caused by the newly-created scattering bars.
- Modifies the scattering bars as needed to eliminate those errors.
- Checks for potential manufacturing and printing problems and cleans and/or trims those scattering bars that would cause problems.

Use of this statement depends heavily on your knowledge of mask preparation requirements and the use of scattering bars. See the *Calibre OPCsbar User's Manual* for detailed information on scattering bar theory and the use of this operation.

**Rule types** — There are two *rule* syntaxes used. [Figure 4-237](#) illustrates SPACE and SPACELAYER rules.

**Figure 4-237. Rule Types**

**Table 4-36** is a summary of the arguments used in this operation. All values are floating-point.

**Table 4-36. Summary of OPCsbar Parameters and Defaults**

Parameter	Type	Default	Description
ANGLED	keyword	Only orthogonal scattering bars allowed	45-degree edges allowed or not.
ANGLEEDGE	keyword	No default	Applies the present rule only to edges derived from angled features.
CENTER	keyword optional, positive, real number	No default	Creates a scattering bar exactly between the two original edges. OR if value is specified, creates two scattering bars, each offset from the center by the value.
CENTERMERGE	positive real number	Do not merge	Merges two scattering bars into one if they are closer than value.
CENTERPITCH	keyword, optional, positive, real number	No default	Creates a scattering bar relative to the center line of the pitch of two contributing features.
COLINEARIZE	positive real number	Do not colinearize	Indicates that <i>overlapping</i> scattering bars that are offset from one another are merged if at least one is shorter than value.
EXTENSION	non-zero real number	Not extended	Edge extended or shortened after creating scattering bars.
HORIZONTAL	keyword	No default	Applies the present rule only to edges derived from horizontal features.
INTERSECTION	Literal: N   L   T  P	L allowed	Types of intersections allowed between scattering bars.
JOG FILL	keyword and positive real number(s)	Jog is cut, not filled	Indicates jog is filled or cut. When value is supplied, fills jogs between nearby (not touching) scattering bars when closer than value.

**Table 4-36. Summary of OPCsbar Parameters and Defaults (cont.)**

<b>Parameter</b>	<b>Type</b>	<b>Default</b>	<b>Description</b>
LINEENDMERGE	positive real number	Do not merge	Merge adjacent scattering bars if they are perfectly aligned and if the distance between them is less than value.
LINEENDOFFSET	non-negative real number	MINSBOFFSET	Minimum separation between the line-end of one scattering bar and the edge of an original feature.
LINEENDTOLONGSPACE	positive real number	LINEENDSPACE	Minimum spacing requirement between the line-end edge of one scattering bar and the long edge of another scattering bar.
LINEENDSPACE	positive real number	MINSBSPACE	Minimum separation between the line-ends of two scattering bars.
MAXSBLENGTH	positive real number	No default	Maximum length for all negative scattering bars.
MAXSBWIDTH	positive real number	Largest SBWIDTH+2	Largest allowed size for final scattering bar.
MINEDGELENGTH	positive real number	0	Minimum length of classified edges for which scattering bars will be generated.
MINJOGWIDTH	positive real number	MINSBWIDTH	Defines the minimum overlap required for a jog to be legal.
MINSBLENGTH	positive real number	Smallest SBWIDTH	Minimum length of scattering bar outside jogs.
MINSBOFFSET	positive real number	Smallest SBOFFSET	Minimum spacing from original polygon to scattering bar.
MINSBSPACE	positive real number	MINSBOFFSET	Smallest allowed distance between scattering bars.
MINSBWIDTH	positive real number	Smallest SBWIDTH-1	Smallest allowed size for final scattering bar.

**Table 4-36. Summary of OPCsbar Parameters and Defaults (cont.)**

<b>Parameter</b>	<b>Type</b>	<b>Default</b>	<b>Description</b>
NEGATIVE	keyword	Not negative — create scattering bars relative to outside edge	Indicates that scattering bars should be created relative to the inside edge of the polygon.
NOCLEANUP	keyword, optional positive integer	Clean and output all scattering bars.	Specifies whether to output raw or cleaned up edges. If cleaned up edges are output, specifies which ones (by priority).
OFFSETLAYER	layer	No default	Defines areas where scattering bars cannot be created. Scattering bars cannot be created inside or within MINSBOFFSET or LINEENDOFFSET of these areas.
OPENT	literal	Trims away the intersection.	Instructs the operation to cleanup T intersections by shortening the stem of the T.
OPPOSITEEXTENDED	non-zero floating point number	No default	Defines the method used to classify edges based on space.
OPTION CAREFUL_CLEAN_UP	keyword, optional	1	Specifies the level that scatterings bars will be recaptured.
OPTION FAST_MRC	keyword, optional	NO (off)	If enabled, FAST_MRC skips the last pass of cleanup which cleans MRC residues in the design
OPTION PRIORITIZE_SPACE	keyword, optional	NO (off)	If enabled, PRIORITIZE_SPACE prioritizes negative scattering bars based on the SPACE condition in a rule.
OPTION VERIFICATION_MODE	keyword, optional	NO (off)	Finds the deleted scattering bars of an executed OPCsbar operation.
PRIORITIZE BY LAYER	layer	No layers: Prioritization by Order	Prioritization scheme for resolving conflicts based on island layers.
PRIORITYCENTER	keyword	No default	Gives the center scattering bar the highest priority during the cleanup process.

**Table 4-36. Summary of OPCsbar Parameters and Defaults (cont.)**

<b>Parameter</b>	<b>Type</b>	<b>Default</b>	<b>Description</b>
SBLAYER	layer	No default	Adds a polygon layer containing pre-existing scattering bars to be factored into calculations when generating additional scattering bars.
SBOFFSET	positive real number	REQUIRED — no default	Creates a scattering bar offset from each original edge by value.
SBPITCH	positive real number	REQUIRED — no default	Creates a scattering bar the distance between the scattering bar center-line and the center of the feature at the nearest valid edge.
SBWIDTH	positive real number	REQUIRED — no default	Width of scattering bar to create.
SMOOTH	positive floating point number	Do not smooth	Smooths out features created by edge classification.
SPACE	constraint: range of real numbers	REQUIRED — no default	Spacing for edge classification. Not used with SPACELAYER
SPACELAYER	layer	No default	Layers identifying edges to receive scattering bar treatment. Not used with ref_layer, SPACE, or CENTER.
STRICTCENTER	keyword	A scattering bar is generated between both edges.	Determines if a scattering bar is generated if the SPACE condition is met and if only one WIDTH condition is met.
V1	keyword	V2	Controls whether V1 or V2 mode is executed.
VERTICAL	keyword	No default	Applies the present rule only to edges derived from vertical features.

**Table 4-36. Summary of OPCsbar Parameters and Defaults (cont.)**

<b>Parameter</b>	<b>Type</b>	<b>Default</b>	<b>Description</b>
VIALAYER	literal	Never remove higher-priority scattering bars to preserve lower-priority scattering bars.	Trims away some of the higher-priority scattering bar when doing so makes it possible for both scattering bars to remain after cleanup.
WIDTH	constraint: range of real numbers	No default	Width for edge classification. The polygon must be this wide to receive scattering bar treatment. Not used with SPACELAYER.
WITHWIDTH	literal	Use INTERNAL OPPOSITE	Classifies edges using With Width.

**Clean-up notes** — Clean-up in OPCsbar involves several steps of adding and removing scattering bars, or portions of them. The order in which these steps occurs can affect the final results. The following clean-up steps occur in this order:

1. COLINEARIZE
2. LINEENDMERGE
3. JOG FILL of touching bars
4. JOG FILL of adjacent bars
5. EXTENSION

## Examples

```
// Create scattering bars for gate layer

SB = OPCSBAR GATE
SPACE >0.3 <= 0.4 SBWIDTH 0.04 CENTER
SPACE >0.4 <= 0.5 SBWIDTH 0.02 SBOFFSET 0.1
SPACE >0.5 <= 0.6 SBWIDTH 0.01 SBOFFSET 0.1
                           SBWIDTH 0.01 SBOFFSET 0.3
SPACE >0.6           SBWIDTH 0.01 SBWIDTH 0.1
                           SBWIDTH 0.01 SBOFFSET 0.3
                           SBWIDTH 0.01 SBOFFSET 0.5

// MINSBOFFSET = 0.1, MINSBSPACE = 0.1, MINSBWIDTH = 0.01,
// MAXSBWIDTH = 0.04, MINSBLENGTH = 0.01,
// L-shaped intersections allowed, LINEENDOFFSET = 0.1,
// LINEENDSPACE = 0.1, no line-end merging, no angled,
// no jog filling, no extension, positive bars,
// default prioritization of layers
```

## OR

Layer operation

**OR *layer1***

**OR *layer2* *layer3* [*layerN...*]**

### Parameters

- ***layer1***  
A required original layer or layer set.
- ***layer2***  
A required original layer or layer set, or a derived polygon layer.
- ***layer3***  
A required original layer or layer set, or a derived polygon layer.

### Description

Merges all polygons on the input layers into one layer. The one-layer operation is equivalent to the [AND \*layer1\* >= 1](#) operation. A two-layer operation generates output equivalent to the polygon data on *layer3* if *layer2* is empty and vice versa. More than two layers are permitted.

Overlapping original polygons are automatically merged prior to evaluating an operation whose input layers are original layers. The merged original polygons are essentially the output of a one-layer OR operation. Therefore, you should rarely need to specify the one-layer OR operation.

Nested or chained OR operations such as this:

```
x = a OR (b OR (c OR ...
```

can have detrimental performance impacts. It is better to use this instead:

```
y = OR a b c ...
```

[DFM Copy](#) also works for this purpose. [Extent Drawn](#) can be useful in the context of finding the extent of the union of many layers.

See also [OR Edge](#), [NOT](#), and [XOR](#).

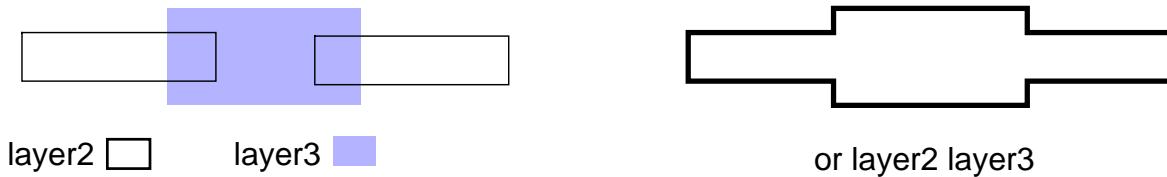
## Examples

The two-layer Boolean OR operation shown in Figure 4-238 merges all overlapping layer2 and layer3 polygons into one polygon. Reversing layer2 and layer3 in the Boolean OR operation shown in Figure 4-238 does not produce different output because Boolean OR is commutative. For example,

**or layer3 layer2**

produces the same output shown in Figure 4-238.

**Figure 4-238. Two-Layer Boolean OR**



## OR Edge

Layer operation

**OR EDGE** *layer1 layer2*

### Parameters

- *layer1*  
A required derived edge layer.
- *layer2*  
A required derived edge layer.

### Description

Performs a Boolean OR on edges from the input layers. This operation is a non-node-preserving layer constructor.

The definition of OR Edge is as follows. This operation:

$z = \text{OR EDGE } x \cup y$

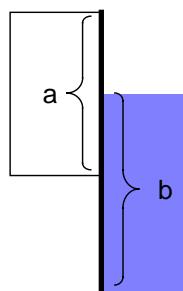
is functionally equivalent to this:

```
a = x NOT COINCIDENT EDGE y
b = y NOT COINCIDENT OUTSIDE EDGE x
z = a ∪ b // the union of a and b
```

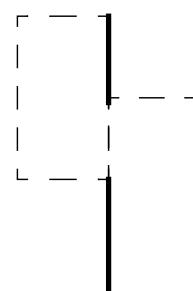
Note that OR Edge does not output coincident edge segments as shown in [Figure 4-239](#):

**Figure 4-239. OR Edge**

- layer1
- layer2
- results



$a = \text{layer1 TOUCH EDGE layer2}$   
 $b = \text{layer2 TOUCH EDGE layer1}$



$a \text{ OR EDGE } b$

To return coincident edge segments in addition to non-coincident ones, use [DFM Copy](#) instead of OR Edge.

See also [OR](#), [Coincident Edge](#), [Not Coincident Edge](#), [Not Coincident Outside Edge](#).

**Example**

The following example shows a layer derivation of ngate and pgate edges using OR Edge.

```

ndiff = diff AND nplus
pdiff = diff AND pplus
gate = poly and diff

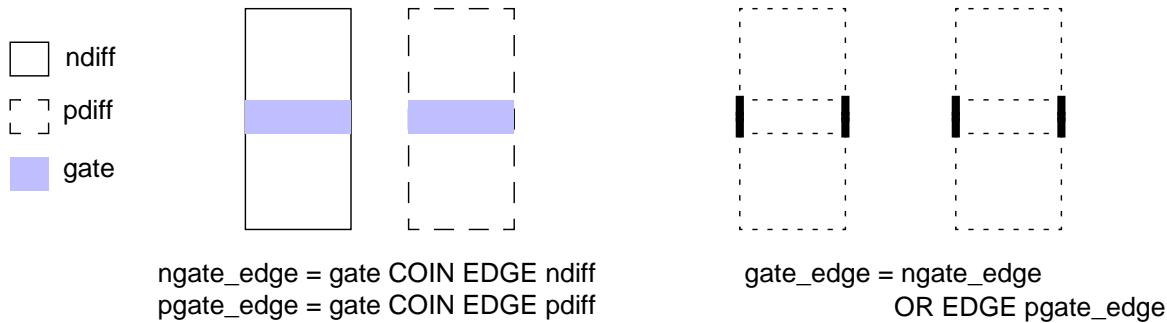
ngate_edge = gate COIN EDGE ndiff
pgate_edge = gate COIN EDGE pdiff

gate_edge = ngate_edge OR EDGE pgate_edge

```

[Figure 4-240](#) shows the derived edge layer gate\_edge from the above example. For clarity, the original layers diff, nplus, and pplus are not shown in the figure.

**Figure 4-240. Gate Edges**



# Ornet

Layer operation

**ORNED *layer1* *layer2* [BY NET | BY SHAPE]**

## Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- ***layer2***  
A required original layer or layer set, or a derived polygon layer.
- **BY NET**  
An optional keyword that instructs the tool to merge overlapping ***layer1*** and ***layer2*** polygons that are on the same net. This is the default behavior if you do not include BY NET or BY SHAPE in the statement.
- **BY SHAPE**  
An optional keyword that instructs the tool not to merge any overlapping ***layer1*** and ***layer2*** polygons on the same net, but to output individual polygons instead.

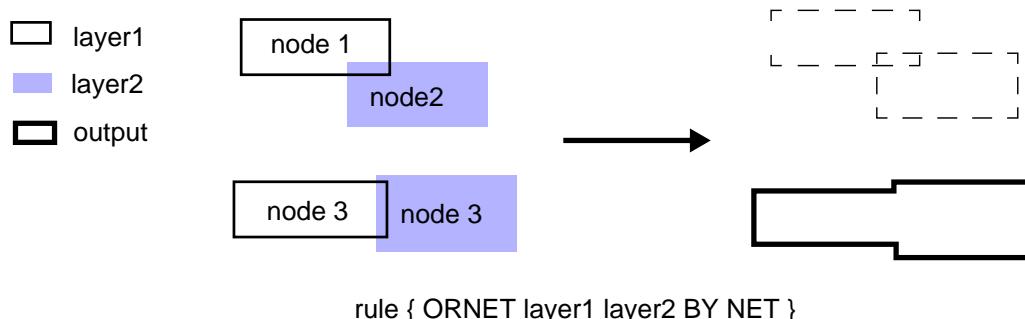
## Description

Performs a Boolean OR operation on all overlapping ***layer1*** and ***layer2*** polygons on the same net. This is the default behavior. Generates output equivalent to the polygon data on ***layer2*** if ***layer1*** is empty and vice versa.

**Requirements and Restrictions** — [Net Area](#), [Net Area Ratio](#), and other Ornet operations having the same settings are the only operations that can have an input layer derived by an Ornet operation. The rule file compiler checks this restriction. This restriction is in place because Ornet can generate *unmerged* data. However, a layer derived by Ornet can be output to the nmDRC results database, which should be its primary use.

The connectivity on the input layers must be established.

## Examples



## Outside

Layer operation

**OUTside *layer1* *layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.

### Description

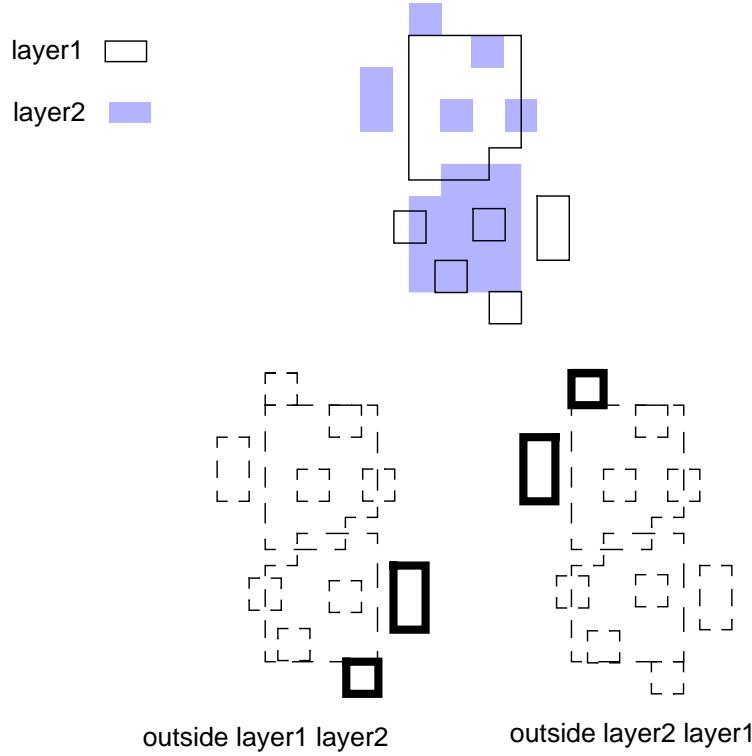
Selects all *layer1* polygons with areas that lie completely outside all *layer2* polygons. A *layer1* polygon that has coincident edges with *layer2* polygons, but does not share area, satisfies the Outside condition. If *layer2* is empty, *layer1* polygons are output.

See also [Not Outside](#), [Outside Edge](#), [Not Interact](#), and [Not Inside](#).

### Examples

Figure 4-241 shows two Outside operations. The operation on the left selects all layer1 polygons that lie outside all layer2 polygons. The operation on the right reverses the layers and selects layer2 polygons that lie outside all layer1 polygons.

**Figure 4-241. Outside**



## Outside Edge

Layer operation

**OUTside EDGE *layer1* *layer2***

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon layer.

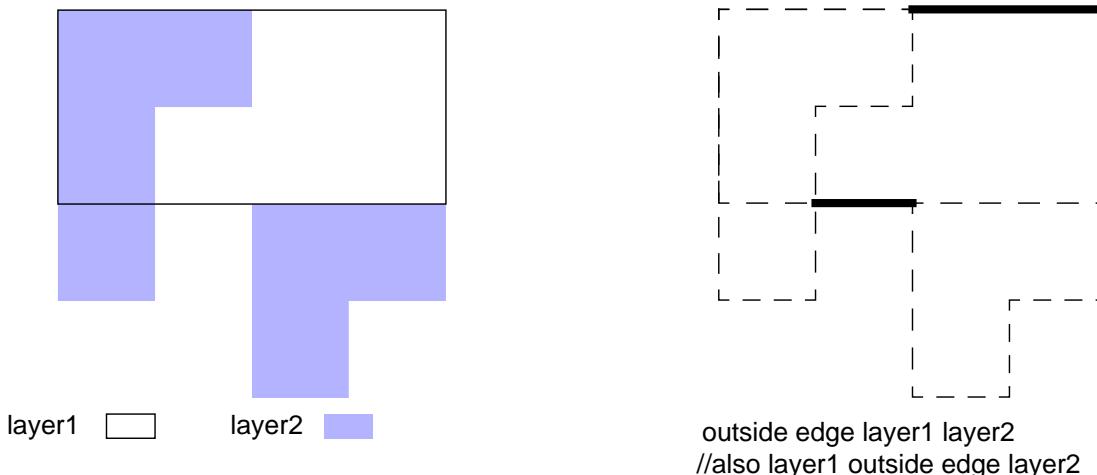
### Description

Selects all *layer1* edges or edge segments that lie completely outside *layer2* polygons. Edge coincidence does not satisfy the Outside Edge condition. If *layer2* is empty, the Outside Edge operation generates output equivalent to the edge data on *layer1*. See also [Not Outside Edge](#), [Not Inside Edge](#), [Coincident Outside Edge](#), [Touch Outside Edge](#), and [Outside](#).

### Examples

The Outside Edge operation shown in Figure 4-242 outputs all *layer1* edges or segments that lie completely outside *layer2* polygons, even if the endpoint(s) of the *layer1* edges or segments touch the outside of the *layer2* edge(s).

**Figure 4-242. Outside Edge**



# Parasitic Variation

Parasitic extraction

Table syntax:

**PARASITIC VARIATION** *layer* *property* {*rule* [*rule*...]}

Thickness syntax:

**PARASITIC VARIATION** *layer* **THICKNESS** *density1* *density2*

## Summary

This optional statement allows you to modify nominal resistance calculations to more accurately model your fabrication process and account for in-die variations.

## Parameters

- ***layer***

A required layer having its nominal resistance values modified.

- ***property***

A required keyword indicating the nominal resistance value which the rules modify.  
Possible choices are:

**SHEET** — A required keyword that specifies the resistance calculation is based upon resistance per-square of the *layer*.

**WIDTH** — A required keyword that specifies the resistance calculation is based upon *layer* polygon width variation.

**RHO** — A required keyword that specifies the resistance calculation is based upon bulk resistivity of the *layer*. Used in conjunction with **THICKNESS**.

**TC1** — A required keyword that specifies the rules override the TC1 coefficient used in [Resistance Sheet](#).

**TC2** — A required keyword that specifies the rules override the TC2 coefficient used in [Resistance Sheet](#).

- **THICKNESS**

A required keyword that specifies the resistance and capacitance calculations are based upon *layer* polygon thickness variation. Used in conjunction with **RHO** for resistance, and used with PEX capacitance rules with thickness for capacitance.

- ***density1* *density2***

Thickness coefficients, used with the **THICKNESS** keyword only. Specify the constant parameters for density-based thickness calculation in the equation:

$$\text{thickness} = \text{density1} * \text{local\_density} + \text{density2}$$

The order of the two parameters is important; they must be entered in the Parasitic Variation THICKNESS statement in the same order they appear in the equation above.

- ***rule***

A required, user-defined rule modifying the default sheet resistance or actual width. You must include at least one ***rule***; multiple rules are allowed. The order of rule parameters is important and must appear as shown in the following format:

**SPACE *constraint* DRAWN\_WIDTH *constraint* VALUE *value***

where:

- **SPACE *constraint*** — Required keyword and constraint defining the **SPACE** condition between a polygon and its nearest neighbor that must be met in order for the ***rule*** to apply. The constraint must be expressed as an interval of floating-point values representing user values, for example,  $> a < b$ . The constraint must be a positive number, must not be a strict equality (as in  $= a$ ), and must not overlap other ranges, or the rule file compiler will report an error. An unbounded interval is interpreted as the bounded complement of the unbounded interval. For example,  $> a$  is interpreted as everything not in the  $\leq a$  interval.

When used in a rule for **TC1** or **TC2**, the **SPACE *constraint*** is ignored and should be set to **SPACE > 0**.

- **DRAWN\_WIDTH *constraint*** — Required keyword and interval constraint in floating-point user units that specifies the width a polygon feature must have in order for the ***rule*** to apply to an edge on that polygon feature. The interval constraint behavior is the same as for the **SPACE** constraint.
- **VALUE *value*** — Required keyword and value defining the resistance of a net with a **SPACE** and **DRAWN\_WIDTH** that meet the rule specifications. For a Parasitic Variation SHEET statement, ***value*** is defined as a non-negative real number, in units of resistance per square, that specifies the proportionality constant. For a Parasitic Variation WIDTH statement, ***value*** is defined, not as the actual width, but as the *delta* from the drawn to the actual width.

## Description

---

### Note



These values are set based on foundry data. Changing the value will affect accuracy.

This statement allows you to modify nominal resistance calculations to more accurately model your fabrication process and account for in-die variations.

---

### Note



The nominal sheet resistance is still specified in the Resistance Sheet SVRF rule. Specifying Parasitic Variation without a corresponding Resistance Sheet causes an error. All other nominal data is specific to the design, and is calculated during extraction.

## Parasitic Variation

---

For a description of the Calibre xRC in-die variation process, including information on when and when not to use it, refer to the [Calibre xRC User's Manual](#).

### Examples

```
PARASITIC VARIATION metall1 WIDTH
    SPACE <= 0.13 DRAWN_WIDTH > 0.11 <= 0.14 VALUE 0.0015
    SPACE > 0.15 <= 0.18 DRAWN_WIDTH > 0.112 <= 0.16 VALUE 0.0235

PARASITIC VARIATION metall1 SHEET
    SPACE <= 0.13 DRAWN_WIDTH > 0.11 <= 0.14 VALUE 0.104
    SPACE > 0.15 <= 0.18 DRAWN_WIDTH > 0.112 <= 0.16 VALUE 0.103

PARASITIC VARIATION metall1 RHO
    SPACE <= 0.08 DRAWN_WIDTH <= 0.08 VALUE 0.0468
    SPACE > 0.08 <= 0.12 DRAWN_WIDTH <= 0.08 VALUE 0.0458

PARASITIC VARIATION metall3 TC2
    SPACE > 0 DRAWN_WIDTH <= 0.08 VALUE 0.3
    SPACE > 0 DRAWN_WIDTH <= 0.12 VALUE 0.4
    SPACE > 0 DRAWN_WIDTH > 0.12 VALUE 0.3

PARASITIC VARIATION metall1 THICKNESS -0.031900 0.205400
```

# Path Length

Layer operation

## PATH LENGTH *layer constraint*

### Parameters

- *layer*

A required derived edge layer.

- *constraint*

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

The constraint must contain non-negative real numbers. It is interpreted in user units.

### Description

Selects a maximum-length contiguous path of *distinct edges* from a single polygon on the input *layer*. The path has a total length that satisfies the *constraint*. Each path of edges is constructed to trace the perimeter of the original polygon. A *layer* edge will not be contained in more than one path because layer data is merged before presentation to this operation.

In certain rare instances, Path Length output in hierarchical mode can differ from output in flat mode. This can occur where polygon corners touch at a singularity. This situation is discussed under “[Path Length Variances](#)” in the *Calibre Verification User’s Manual*.

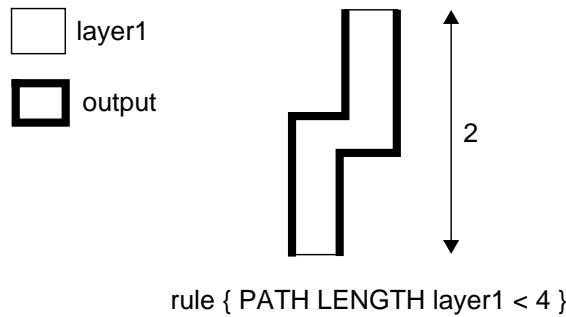
You should be careful when specifying constraints in the Path Length operation. A potential problem arises when you make assumptions about the exact length of paths. For example, you might want to avoid using constraints such as == 5 when layer1 contains edges that are not orthogonal to the database axes. A better choice of constraint is > 4.995 < 5.005. This allows a tolerance of ten database units, if your [Precision](#) is 1000.

See also [Length](#).

### Examples

#### Example 1

**Figure 4-243. Path Length**



rule { PATH LENGTH layer1 < 4 }

In this case, the output consists  
of six edges.

#### Example 2

This example shows a rule check that finds gates (including bent ones) whose total width, measured along the gate edges, is less than 4 um.

```
gate = poly AND diff

gate_edge = gate COIN EDGE poly
short_edge = gate_edge PATH LENGTH < 4
// short_edge can have paths containing multiple edges

narrow_gate {
    @ Gate width, including that for bent gates, must be at least 4 microns.
    gate WITH EDGE short_edge
}
```

## Pathchk

Layer operation

```
PATHCHK {labeled_flag | {power_ground_flag [operand power_ground_flag]}} [layer]
[PRINT POLYGONS filename [BY LAYER] [BY CELL | FLAT]]
[PRINT NETS filename]
[PORTS ALSO] [NOFLOAT] [EXCLUDE UNUSED]
[EXCLUDE SUPPLY]
[POWERNAME name [name ...]]
[GROUNDNAME name [name ...]]
[LABELNAME name [name ...]]
[BREAKNAME { break_net | cell_name '(' break_net [break_net ...] ')' } ...]
```

Used during circuit extraction in Calibre nmLVS/nmLVS-H and ICtrace Mask mode.

### Summary

This operation is used for ERC rule checks. It selects polygons from nets in the layout that do or do not have a path to nets that meet specified conditions. Power, ground, and labeled nets are top-level nets. This is performed as a part of connectivity extraction in LVS. Used in conjunction with [ERC Select Check](#), [ERC Unselect Check](#), and [LVS Execute ERC](#).

### Parameters

- *labeled\_flag*

Keyword that specifies how the tool outputs polygons on nets that do or do not have a path to a labeled net. This option uses top-level text objects that are neither power nor ground net labels. Typically these will be port names. Possible choices are:

**LABELED** — Report unlabeled nets with a path to a labeled net.

**! LABELED** — Report unlabeled nets with *no* path to *any* labeled net.

where the exclamation mark (!) is synonymous with Boolean NOT.

- *power\_ground\_flag*

Keyword that specifies how the tool outputs polygons on nets that do or do not have a path to a power or ground net. Power and ground nets are top-level nets specified in the [LVS Power Name](#) and [LVS Ground Name](#) statement. Possible choices are:

**POWER** — Reports nets with a path to power.

**! POWER** — Reports nets with no path to power.

**GROUND** — Reports nets with a path to ground.

**! GROUND** — Reports nets with no path to ground.

Where the exclamation mark (!) is synonymous with Boolean NOT.

The *power\_ground\_flag* keyword when not combined by an operand is a *primitive* condition.

- *operand*

An optional logical symbol that combines two *power\_ground\_flag* conditions. Possible choices are:

&& — Selects nets that satisfy both primitive conditions: Boolean AND.

|| — Select nets that satisfy the first or second primitive condition, or both: Boolean OR.

- *layer*

An optional layer name that directs the Pathchk operation to select polygons only from the specified layer. This layer must appear in a [Connect](#) or [Sconnect](#) statement. When you do not specify the *layer* parameter, the Pathchk operation selects and merges polygons from all layers specified in Connect and Sconnect statements.

PRINT parameters are not affected by the *layer* parameter.

- PRINT POLYGONS *filename* [BY LAYER] [BY CELL | FLAT]

An optional keyword followed by a pathname that instructs the tool to generate polygon output of all nets that satisfy the Pathchk condition. The *filename* specifies where the tool writes the generated polygon file in nmDRC ASCII format. By default, all results from each Pathchk operation are written to one check in the result file.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

The following keywords may be specified with PRINT POLYGONS; they have no effect on PRINT NETS output:

BY LAYER — An optional keyword that classifies PRINT POLYGONS results by layer. The tool creates a separate check for each layer; the check contains nets reported for the layer. All results are reported in top-level coordinate space.

The tool generates a check for each layer in each cell when you specify BY LAYER and BY CELL in the same statement.

BY CELL — An optional keyword that classifies PRINT POLYGONS results by cell. The tool creates a separate check for each cell that has cell-specific results; the check contains nets reported in that cell. All results are reported in top-level coordinate space.

Flat applications ignore this keyword.

FLAT — An optional keyword that specifies PRINT POLYGON results to be flat; cell-specific results are replicated in every placement. This keyword has no effect in flat applications and cannot be specified with BY CELL.

Creation of empty rule checks by PRINT POLYGONS is controlled by the [ERC Keep Empty](#) statement. The output *filename* is not created if all rule checks are empty and ERC Keep Empty NO is specified.

- PRINT NETS *filename*

An optional keyword followed by a pathname that instructs the tool to generate output of all nets that satisfy the Pathchk condition. The *filename* specifies where the tool writes the generated text file.

In hierarchical execution, the report file format is this:

```
PATHCHK REPORT for <layout primary>
PATHCHK <condition>:
  cell <name> (<placement>): net1, net2, ...
  cell <name> (<placement>): net1, net2, ...
```

where *condition* is the Pathchk condition (for example, GROUND && ! POWER), *name* is a cell name where results were found, *placement* is a representative placement of that cell, and *netN* is a net name or number if the net has no name. Cells are listed in bottom-up order. Net names and numbers are in the context of the indicated cells. Cells for which no nets are found by Pathchk are not written to the report file.

In flat execution, the report file format is this:

```
PATHCHK REPORT for <layout primary>
PATHCHK <condition>:
  net1, net2, ...
```

If for a particular request no nets are found (in any cell), nothing is written out for that request. If all requests generate empty results, the report file is not written.

- PORTS ALSO

An optional keyword that instructs the tool to include nets that are connected to port objects in the top cell in the Pathchk output. By default, nets that are connected to ports in the top cell are not considered by Pathchk. This option affects both PRINT POLYGONS and PRINT NETS output. In Calibre, ports are indicated using [Port Layer Text](#); in Pyxis Layout, ports are indicated with \$make\_port().

- NOFLOAT

An optional keyword that instructs the statement to ignore floating nets. Floating nets are nets that are not connected to any devices.

- EXCLUDE UNUSED

An optional keyword that instructs the statement to exclude *unused* nets from the ERC Pathchk output. Unused nets are connected only to unused devices. Unused devices are specified with [LVS Filter Unused Option](#) or equivalent LVS Filter Unused statements. All options for LVS Filter Unused Option are supported except the INV, P, and Q options.

- EXCLUDE SUPPLY

An optional keyword that instructs the tool to exclude power and ground nets from the results of a Pathchk operation containing at least one of the keywords **POWER** or **GROUND** (and not preceded by the ! operator). Supply nets are included if EXCLUDE SUPPLY is not specified.

- **POWERNAME** *name*

An optional keyword and parameter that specifies which nets should serve as power nets in this particular Pathchk operation. It overrides the [LVS Power Name](#) list in this particular operation. This option can only be specified when the **POWER** keyword is used. More than one *name* may be specified.

- **GROUNDNAME** *name*

An optional keyword and parameter that specifies which nets should serve as ground nets in this particular Pathchk operation. It overrides the [LVS Ground Name](#) list in this particular operation. This option can only be specified when the **GROUND** keyword is used. More than one *name* may be specified.

- **LABELNAME** *name*

An optional keyword and parameter that specifies which nets should be treated as labeled in this particular Pathchk operation. This option can only be specified when the **LABELED** keyword is used. More than one *name* may be specified.

The string `%_p_` can be used as a *name* to specify all ports in the top cell. The pattern `%_p_` is replaced by the string containing the names of all top-level ports.

- **BREAKNAME** { *break\_net* | *cell\_name* ‘( *break\_net* [*break\_net* ...] )’ } ...

An optional keyword and parameter group that specifies the names of labeled nets that break paths. The net names can be specified for individual cells. The parameters are as follows:

*break\_net* — A name of a net that breaks paths to supply nets or labeled pads. If the only path to some net N from a given signal goes through at least one *break\_net*, then N is considered as not having a path to the supply nets or labeled pads. If specified without a *cell\_name* parameter, the *break\_net* is considered to be a top-level net and breaks paths in all instantiated hierarchy. A list of *break\_net* names may be specified. The “?” wildcard character matches zero or more characters in a net name.

*cell\_name* — A name of a cell in which specified *break\_net* names are found. When *cell\_name* is specified, the *break\_net* names associated with it must appear in a space-delimited list in parentheses. In this case, a *break\_net* has the effect of breaking paths in the cell in which it is specified, and in all cells placed in that cell. However, no breaking occurs on nets connected to a *break\_net* at a higher level of hierarchy. More than one *cell\_name* group may be specified. No wildcards may be used for cell names in this case. Cell names are always case-insensitive.

See [Example 6](#) for one use of the BREAKNAME keyword.

## Description

This operation constructs layers consisting of layout nets that do or do not have a path to top-level power nets, ground nets, labeled nets, or nets that satisfy a combination of these conditions. This layer operation is not node-preserving.

**Note**

You cannot use a derived layer from a Pathchk operation in [Connect](#), [Sconnect](#), or [Device](#) operations, nor use such a layer to derive other layers that appear in these operations.

By default, a *path* is any route that leads through source/drain pins of built-in MOS devices or pos/neg pins of built-in resistor devices. To qualify, a MOS device (types M, ME, MD, MN, MP, or any of the LDD-type devices) must have at least the following pins: G (or gate), S (or source), and D (or drain). Paths through LDD-type devices are formed in both directions. Capacitor, diode, or bipolar devices can also be added to the path, as specified in [ERC Path Also](#) statements.

If you want to enable user-defined devices to participate in Pathchk (user-defined devices do not participate by default), then you must map these devices and their pins to built-in devices and pins. This is done using the [LVS Device Type](#) specification statement. See “User-Defined Devices in Path Check Operations” in the [Calibre Verification User’s Manual](#).

Power and ground nets are specified with [LVS Power Name](#) and [LVS Ground Name](#), respectively. Power and ground nets break paths.

Labeled nets are those with text, but they are not power or ground nets. Labeled nets typically belong to top-level I/O ports.

The hierarchical level from which text objects are read is controlled by the [Text Depth](#) statement.

Pathchk respects the [LVS Compare Case](#) statement setting when tracing power and ground net names. If either the LVS Compare Case YES or LVS Compare Case NAMES statement is specified, then power names and ground names are traced in a case-sensitive manner; otherwise they are case insensitive. These rules on case sensitivity apply to LVS Power Name and LVS Ground Name nets, and to names specified in POWERNAME, GROUNDNAMES, LABELNAME, and BREAKNAME options.

The search features [Find Path](#) and [Isolate Path on Layout Net](#) in Calibre RVE for LVS may also be of interest.

## Supply Name Behaviors

Power and ground nets are included in the result for the keywords **POWER** and **GROUND**. For example, supply nets are included if you specify the following in a rule check:

### **PATHCCHK POWER**

You can exclude supply nets by specifying EXCLUDE SUPPLY. When specifying EXCLUDE SUPPLY, at least one of the **POWER** or **GROUND** keywords must also be specified in the operation. (The EXCLUDE SUPPLY keyword does not apply to the ! **POWER** or ! **GROUND** keywords.) For example:

### **PATHCCHK POWER && !GROUND EXCLUDE SUPPLY**

is allowed. However, this syntax causes a compiler error:

### **PATHCCHK ! POWER && ! GROUND EXCLUDE SUPPLY // bad syntax**

Power and ground nets are not included in the result for **! POWER** and **! GROUND** keywords. For example, the supply net itself is not selected by this:

**PATHCHK ! GROUND**

even if there is no path from power to ground.

If a net in the design matches more than one of LVS Power Name, LVS Ground Name, Pathchk POWERNAME, Pathchk GROUNDNAMES, or Pathchk LABELNAME name patterns, a warning is issued and the conflicts are resolved according to the following priority rules:

- Pathchk keyword overrides are given priority over LVS Power Name or LVS Ground Name settings.
- Power names are given priority over ground names.

The **LABLED** option does not consider supply nets as labeled.

Pathchk observes the [Layout Preserve Case](#) specification statement for case-sensitivity of net names. See “[Case Sensitivity of Power and Ground Names](#)” in the *Calibre Verification User’s Manual* for more details about power and ground names.

## BREAKNAME Behavior

BREAKNAME nets are special cases of nets that also satisfy the **LABLED** condition. If you perform a Pathchk **LABLED** operation with **BREAKNAME** specified, the **BREAKNAME** nets will appear in any reported results. This is true even if the **BREAKNAME** net is the only texted net in the design that satisfies the **LABLED** condition.

**BREAKNAME** nets that are specified in Pathchk **POWER** or Pathchk **GROUNDS** operations are always reported in any results.

## Execution of Pathchk Operations

Pathchk is used by Calibre nmLVS/nmLVS-H and Mask-mode ICVerify. It is executed during circuit extraction and only when the layout is geometric. You must use [ERC Select Check](#) to enable ERC rule checks in an LVS run. LVS Execute ERC NO disables ERC checks that would otherwise be performed. The following Calibre commands execute this statement:

```
calibre -lvs          // executed flat
calibre -lvs -tl      // executed flat
calibre -lvs -hier    // executed hierarchically and
                      // Layout System is geometric
calibre -spice        // executed hierarchically
```

The following Pyxis Layout commands execute this statement:

```
$lvs_mask()          // executed flat
$write_mask_cnet()   // executed flat
```

Note these important considerations:

- The **POWER**, **GROUND**, and **LABLED** options (without the **!** operator) are not recommended because they can generate very large files, consume large amounts of

memory, and cause long runtimes. In hierarchical execution, such requests flatten the design hierarchy.

- Pathchk operations are not executed if the specified net is not present. For example, if there is no power net in the layout, then Pathchk ! POWER reports an error message and generates no output.
- LVS Power Name and LVS Ground Name nets always break paths regardless of POWERNAME or GROUNDNAMES overrides. In addition, nets specified in POWERNAME and GROUNDNAMES break paths as well.
- Determination of power and ground names for the purpose of the EXCLUDE UNUSED option is controlled exclusively by the LVS Power Name and LVS Ground Name specification statements; the POWERNAME and GROUNDNAMES overrides do not apply.
- The POWERNAME, GROUNDNAMES, and LABELNAME options can be specified in any order, but must be the last options of the Pathchk statement. They work both flat and hierarchically, and they do not apply to [ERC Pathchk](#) statements.
- In Calibre nmDRC/nmDRC-H and ICrules, the Pathchk operation produces an empty result layer and generates a warning, and PRINT options are not executed.
- Pathchk includes floating nets when they fit the specified request and you do not specify the NOFLOAT parameter. For example, the following statement's output includes any floating nets that exist:

#### **PATHCHK ! POWER**

While the next statement's output does not include any floating nets:

#### **PATHCHK ! POWER && GROUND NOFLOAT**

- In the default mode (neither FLAT nor BY CELL specified), the PRINT POLYGONS output may contain polygons belonging to a cell in two ways.
  - If a net in the cell violates the specified requirements in some but not all placements of this cell, the polygons are written out for each placement where the violation occurs, in top-level coordinate space.
  - If, for a given net, the violation occurs in all placements of the cell, the polygons of this net are written out only once in the lowest leftmost placement of the cell.

This could be difficult to interpret, because in the same cell some polygons may be displayed in many placements, while other polygons only appear in one placement.

If the results are difficult to interpret, you can use the BY CELL option to separate cell-specific results in different checks. The FLAT option can also be used to force Calibre to output cell-specific results in all placements. This eliminates any confusion, but increases both memory consumption and the size of the polygon file. Output using the PRINT NETS option is unambiguous.

- Results of the operation are optimized for the different outputs produced. Specifically, the output layer produced from a given input layer preserves hierarchy as much as possible, resulting in more compact data with less promotion. Such operations are most commonly used as part of a more complex SVRF script where the result layer serves as an input layer for other nmDRC operations.
- The output layer of the operation without an input layer parameter specified, which is commonly used for results presentation, is optimized for readability and clarity. For each net selected by Pathchk which is a pin of a cell, all geometries are promoted to the containing cell and reported in its context as a part of the parent net. Output generated by [ERC Pathchk](#) and the PRINT POLYGONS option of Pathchk is optimized in the same way, since it is also intended for results viewing. In no case are the same polygons reported in both child and parent cells.

## Performance Considerations

Pathchk operations that specify the same set of net name overrides (POWERNAME, GROUNDNAMES, LABELNAME) are executed concurrently, that is, as a single operation with multiple output layers. The concurrency of Pathchk operations saves some computations that are the same for all concurrent operations.

For example, if the rule file contains the following Pathchk operations:

```
PATHCHK POWER && GROUND POWERNAME VDD      // 1
PATHCHK POWER && !GROUND POWERNAME VDD      // 1
PATHCHK POWER pgate POWERNAME VDD            // 1
PATHCHK POWER ngate POWERNAME VDD           // 1
PATHCHK POWER && GROUND GROUNDNAMES VSS     // 2
PATHCHK POWER && !GROUND GROUNDNAMES VSS    // 2
PATHCHK POWER pgate GROUNDNAMES VSS         // 2
PATHCHK POWER ngate GROUNDNAMES VSS         // 2
PATHCHK POWER && GROUND                   // 3
PATHCHK POWER && !GROUND                  // 3
PATHCHK POWER pgate                      // 3
PATHCHK POWER ngate                      // 3
```

then three concurrent sets are executed as shown by the comments above. Note that Pathchk operations with no overrides are executed in a separate concurrency set.

Hierarchical Calibre operations impose a size limit on intermediate data structures generated by Pathchk. The limit is approximately 100 MB on 32-bit platforms and 400 MB on 64-bit platforms. It is applied independently to each Pathchk operation. If the limit is reached, Pathchk generates incomplete results (some polygons may be omitted from the output) and issues the following warning:

WARNING: ERC operation exhausted allowed memory allocation.

Calibre issues the warning in both the transcript and the circuit extraction report. This limit applies only to polygon output (that is, construction of a result layer) and the PRINT POLYGONS option. Net reporting from the PRINT NETS option is always complete.

This limit exists to protect against excessive memory usage in rare cases when a very large net is selected by Pathchk (for example, a power or ground net), and cases where a very large part of the design is selected. The limit should not be reached under normal conditions.

Positive requests, like Pathchk POWER, are not recommended because they can generate very large files, consume much memory, and take a long time. In particular, in hierarchical execution such positive requests essentially flatten the design hierarchy.

In certain situations, the Pathchk operation consumes large amounts of memory and time if the operation generates large results, and those results are output directly to the results database. For example, consider a design where the power pad is not properly connected to the actual power net, so that a check like:

```
ERR { PATHCHK ! POWER }
```

attempts to return essentially the entire power routing. This can be very time-consuming, in particular when the power routing has the form of a grid. The result will be a huge, flat polygon consisting of all the power routing layers merged together. In such cases, the Pathchk single-layer mode (using the *layer* argument and writing individual per-layer checks) is likely to reduce resource consumption. Another solution is to use the PRINT NETS option instead of writing the full shapes to the results database. Also, the [ERC Pathchk](#) specification statement (as opposed to Pathchk) is better optimized to handle such cases.

Note that if the Pathchk result is operated upon before being output to the results database, then resource consumption may not be an issue. For example, the following is likely to work, because the final amount of data written to the results database is reduced:

```
A = PATHCHK ! POWER
ERR { A AND CONTACT }
```

## Examples

### Example 1

The following example creates a derived layer named erc\_layer containing a representation of all nets with no path to POWER.

```
erc_layer = PATHCHK ! POWER
```

### Example 2

The following example creates a file named file1 that containing polygon output of all nets with no path to POWER.

```
erc_rule {PATHCHK ! POWER PRINT POLYGONS "file1"}
```

### Example 3

When not using the [!] LABELED option, the following combinations are possible:

**! POWER && GROUND**

Nets with no path to power but with a path to ground

**! GROUND && POWER**

Nets with no path to ground but with a path to power

**! GROUND && ! POWER**

Nets with no path to ground and no path to power

**! GROUND || ! POWER**

Nets with no path to ground or no path to power (or both)

**POWER && GROUND**

Nets with a path to both power and ground

**POWER || GROUND**

Nets with path to power or a path to ground

#### **Example 4**

This shows the use of the **! LABELED** option to print nets without labels:

```
rule { PATHCHK !LABELED PRINT NETS unlabeled_nets.txt }
// show nets with no path to any labeled net
```

#### **Example 5**

This example shows a check for gates that are not connected either to power or ground.

```
CONNECT gate poly

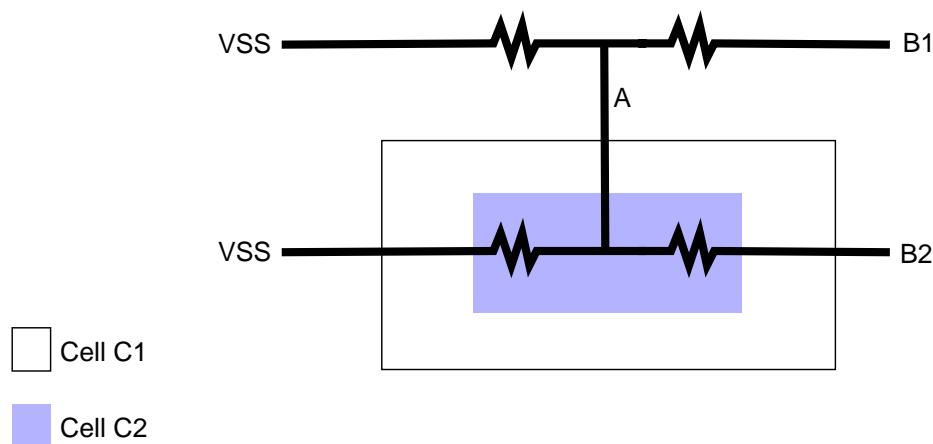
gate_not_pwr_or_gnd { PATHCHK !POWER && !GROUND gate }

LVS EXECUTE ERC YES
ERC SELECT CHECK gate_not_pwr_or_gnd
```

#### **Example 6**

This example illustrates the use of the **BREAKNAME** keyword. In [Figure 4-244](#), net B1 has a path to the ground signal VSS through two resistors in the top cell. Net B2 has a path to VSS through two resistors in cell C2.

**Figure 4-244. Pathchk BREAKNAME**



You can use Pathchk to find all nets with a path to ground (VSS), and designate net A as a breaking net in cell C1:

```
C1_break {PATHCHK GROUND BREAKNAME C1(A)}
```

In this case, net B1 has a path to ground because the breaking net A in cell C1 has no effect outside of cell C1. However, net B2 does not have a path to ground because net A breaks paths in cell C1, and all contained cells, including cell C2.

## PERC Load

Specification statement

**PERC LOAD *tvf\_function* [XFORM {REDUCTION | INJECTION | ALL}]  
[INIT *init\_proc*] SELECT *check\_proc* [*check\_proc* ...]**

Used only in Calibre PERC applications.

### Summary

Specifies the PERC initialization procedures and rule checks to execute. This statement must be present in the rule file for any PERC rule checks to be performed.

Unlike most SVRF statements, the order of the parameters in this statement is important.

See the [Calibre PERC User's Manual](#) for a complete description.

### Parameters

- ***tvf\_function***

A required parameter that specifies the [TVF Function](#) to be loaded into PERC's embedded Tcl interpreter. All *init\_proc* and *check\_proc* procedures specified in the PERC Load statement must be defined within *tvf\_function*. The parameter *tvf\_function* is case-insensitive.

- **XFORM {REDUCTION | INJECTION | ALL}**

An optional keyword that instructs PERC to perform the indicated netlist transformation before executing any *init\_proc* or *check\_proc* procedures. See [PERC Netlist Transformations](#) for details about the transformation process.

- **INIT *init\_proc***

An optional keyword and parameter that specify a PERC initialization procedure, where *init\_proc* is defined in *tvf\_function* and must be a Tcl proc that takes no arguments. The parameter *init\_proc* is case-sensitive. PERC initialization procedures are described in the “[Initialization Procedures and Commands](#)” chapter of the [Calibre PERC User's Manual](#).

- **SELECT *check\_proc* [*check\_proc* ...]**

A required keyword and parameter that specify a PERC rule check procedure, where *check\_proc* is defined in *tvf\_function*, and must be a Tcl proc that takes no arguments. This keyword and its parameters must appear last in the PERC Load statement.

You can specify any number of rule check procedures. If multiple *check\_proc* parameters are specified, each must have a unique name. PERC rule checks are described in the “[Rule Check Procedures and Commands](#)” chapter of the [Calibre PERC User's Manual](#).

## Description

Selects the PERC initialization procedures and rule checks to execute. This statement must be present in the rule file for any PERC results to be generated.

The PERC Load statement may appear any number of times in the rule file. For each unique combination of *tvf\_function*, *init\_proc* and XFORM choice, there can be at most one PERC Load statement.

Each PERC Load statement is independent of any other in the same rule file. More precisely, PERC follows these steps for each PERC Load statement:

1. Create new Tcl interpreter.
2. Read original netlist.
3. Perform the netlist transformation, if specified.
4. Execute the initialization procedure, if provided.
5. Execute the rules checks in the order listed in the statement.

**Processing Order of PERC Load statements** — The statements are not necessarily processed in the order that they appear in the rule file. Instead, PERC arranges them into four groups based on the netlist transformation selection, and processes them in the following order:

1. No netlist transformation.
2. REDUCTION transformation.
3. INJECTION transformation.
4. ALL transformation.

Within each group, the PERC Load statements are processed in the order that they appear in the rule file.

## PERC Netlist Transformations

PERC netlist transformations occur before any initialization procedures or rule checks are performed. The allowed transformations are those enabled through various nmLVS specification statements either by default or by explicit inclusion in the rule file. The transformations are these:

Device reduction and filtering — as specified by [LVS Reduce](#), the LVS Reduce family of statements, and the [LVS Filter Unused](#) family of statements.

Logic injection — as specified by [LVS Inject Logic](#).

Gate recognition — as specified by [LVS Recognize Gates](#).

The use of the statements is shown in [Example 1](#). Also see [LVS Circuit Comparison](#) in the *Calibre Verification User's Manual* for a discussion of these topics.

The XFORM selections are progressive, as shown in [Table 4-37](#).

**Table 4-37. XFORM Keyword Options for PERC Load**

Keyword	Device Reduction and Filtering	Logic Injection	Gate Recognition
REDUCTION	Yes	No	No
INJECTION	Yes	Yes	No
ALL	Yes	Yes	Yes

Be aware that for any of the LVS Reduce family, LVS Filter Unused family, LVS Inject Logic, or LVS Recognize Gates statements for which you explicitly disable the associated functionality by specifying NO or NONE, PERC Load *will not override* these explicit settings.

If you specify PERC Load ... XFORM ALL, you should normally specify LVS Inject Logic NO if your PERC rule checks depend upon finding normal logic gates in the netlist.

The XFORM keyword should be avoided in PERC LDL applications.

See also [PERC Report](#).

## Examples

### Example 1

Assume the rule file has the following statements:

```
LVS FILTER UNUSED MOS yes
LVS REDUCE SPLIT GATES no
LVS INJECT LOGIC no
LVS RECOGNIZE GATES simple
PERC LOAD ex_func XFORM all SELECT check_w
```

The choice of ALL in the PERC Load statement triggers the following netlist transformations: device reduction, unused device filtering, logic injection, and gate recognition. However, the transformations are done according to the LVS specification statements present in the rule file. In this case, unused MOS devices are filtered out. The structures enabled by the default reduction rules, such as parallel MOS devices, are reduced, but split gates are not reduced. Logic injection is not performed because it is disabled. Finally, simple gates are formed while complex gates are not.

### Example 2

This example demonstrates the execution order for procedures. Assume the rule file has two TVF functions and four PERC Load statements:

```
TVF FUNCTION func_1 /*

    proc setup_1 {} {
        # PERC commands
    }
    proc check_1 {} {
        # PERC commands
    }
```

```

proc check_2 {} {
    # PERC commands
}
proc check_3 {} {
    # PERC commands
}
proc check_4 {} {
    # PERC commands
}
*/
]

TVF FUNCTION func_2 /*

proc setup_a {} {
    # PERC commands
}
proc setup_b {} {
    # PERC commands
}
proc check_a {} {
    # PERC commands
}
proc check_b {} {
    # PERC commands
}
*/
]

PERC LOAD func_1 INIT setup_1 SELECT check_1 check_2
PERC LOAD func_1 XFORM all INIT setup_2 SELECT check_3
PERC LOAD func_1 XFORM reduction INIT setup_1 SELECT check_4
PERC LOAD func_2 XFORM all INIT setup_a SELECT check_b

LVS INJECT LOGIC NO // recommended setting with XFORM all

```

PERC executes the rule checks in this order:

**Table 4-38. Order of Rule Check Execution in PERC**

Procedure	PERC Load statements
setup_1, check_1, check_2	PERC LOAD func_1 INIT setup_1 SELECT check_1 check_2
setup_1, check_4	PERC LOAD func_1 XFORM reduction INIT setup_1 SELECT check_4
setup_2, check_3	PERC LOAD func_1 XFORM all INIT setup_2 SELECT check_3
setup_a, check_b	PERC LOAD func_2 XFORM all INIT setup_a SELECT check_b

## PERC Netlist

Specification statement

### PERC NETLIST {LAYOUT | SOURCE}

Used only in Calibre PERC applications.

#### Parameters

- **LAYOUT**

A required keyword that specifies PERC operates on the layout netlist. This is the default if this statement is not included in the rule file.

- **SOURCE**

A required keyword that specifies PERC operates on the source netlist.

#### Description

Specifies the input design database for PERC. This statement may appear at most once.

When the -layout command-line option is specified, this causes PERC Netlist LAYOUT to be used during the run, regardless of what appears in the rule file.

You can cause PERC to output its internal view of the connectivity by specifying [LVS Write Layout Netlist](#) or [LVS Write Source Netlist](#). The type of output netlist should match either the LAYOUT or the SOURCE keywords, otherwise the output netlist is empty.

See also [PERC Load](#).

#### Examples

##### Example 1

This example assumes that a clean layout netlist extraction has already occurred. PERC is run against the layout netlist by default, as specified by Layout Path.

```
LAYOUT SYSTEM SPICE
LAYOUT PATH "/home/workspace/layout/top.spi"
LAYOUT PRIMARY TOP
...
PERC REPORT "perc.layout.report"
PERC LOAD perc_check INIT make_paths SELECT check1 check2
...
TVF FUNCTION perc_check /*
```

**Example 2**

This example shows the configuration for a PERC run on a source netlist.

```
SOURCE SYSTEM SPICE
SOURCE PATH "/home/designs/spice/top.spi"
SOURCE PRIMARY TOP
...
PERC NETLIST source
PERC REPORT "perc.source.report"
PERC LOAD perc_check INIT name_nets SELECT check1 check2
...
TVF FUNCTION perc_check /*
```

## PERC Pattern Path

Specification statement

### PERC PATTERN PATH *filename*

Used only in Calibre PERC applications.

#### Parameters

- *filename*

A required path name of a pattern file.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

#### Description

This statement specifies the PERC pattern file name. The *filename* must be in SPICE format. Each subcircuit in the file is a pattern template that can be used for pattern matching during rule checking. The `perc::get_one_pattern` command accesses pattern templates from the *filename*.

As a pattern template, each subcircuit can only contain primitive devices with unique names. Since pattern matching is based on netlist connectivity, all devices and nets of a subcircuit must be connected. In other words, for a subcircuit to be a valid pattern template, there must be a path from any node to any other node.

The pattern file must contain a subcircuit named “top”. This subcircuit is not a pattern template, rather it is needed to instantiate the other subcircuits. As a result, the name “top” is not a valid pattern template name.

This statement may appear at most once. If you do not include this statement in the rule file, then pattern matching is disabled.

If you use this statement and you use [PERC Load XFORM ALL](#) or [XFORM INJECTION](#), then you should also set [LVS Inject Logic NO](#) and [LVS Recognize Gates NONE](#). This is to preserve the topology of the original netlist when searching for patterns.

The pattern template names are case-sensitive if the [LVS Compare Case YES](#) or [TYPES](#) has been specified. The pattern device and net names are case sensitive if [LVS Compare Case YES](#) or [NAMES](#) has been specified.

## Examples

This shows a specification of a pattern.

```
PERC PATTERN PATH "pattern.sp"
```

File pattern.sp (contains two pattern templates):

```
.SUBCKT pat1 a vss
M0 a b c vss N
M1 d b e vss N
r0 a vss
.ENDS

.SUBCKT pat2 a b c vss
M0 a b c vss N
M1 d b e vss N
C0 a vss
.ENDS

.SUBCKT top      $ The name must be "top"
x1 1 2 pat1    $ dummy instance
x2 1 2 3 4 pat2 $ dummy instance
.ENDS
```

## PERC Property

Specification statement

**PERC PROPERTY** [STRING] *component\_type* [ ‘( *component\_subtype* ‘) ]  
*property* [*property* ...]

Used only in Calibre PERC applications.

### Parameters

- STRING

An optional keyword that specifies that the properties listed in this statement are string properties. If STRING is not present, then the properties are numeric properties.

- *component\_type*

A required parameter that specifies the device component type. See [Device](#) and “[Built-In Device Types](#)” in the *Calibre Verification User’s Manual* for a discussion of device component types.

The parameter *component\_type* is case-sensitive if the [LVS Compare Case](#) specification statement has been specified with the YES or TYPES parameter.

- ( *component\_subtype* )

An optional parameter, which must be enclosed in parentheses, that specifies the device component subtype. It is also called the model name.

The parameter *component\_subtype* is case-sensitive if the LVS Compare Case specification statement has been specified with the YES or SUBTYPES parameter.

- *property*

A required parameter that specifies a valid SPICE device property. You can specify *property* any number of times in this statement, but each property must be unique. See [Table 4-5](#) for a list of built-in devices with default properties.

Property names are case-insensitive.

### Description

Instructs PERC to read the given list of properties from the input netlist, regardless of whether the properties are needed by nmLVS. These properties are then available for use during PERC rule checking.

By default, PERC only reads device properties that are *actionable* by nmLVS, such as the ones mentioned in [Trace Property](#) or the [LVS Reduce](#) ... TOLERANCE family of statements. The complete description of actionable properties is under “[SEPARATE PROPERTIES Usage](#)” on page 977. Those properties are automatically available for use during PERC rule checking.

The PERC Property statement may appear any number of times in the rule file. For each unique combination of *component\_type*, *component\_subtype*, and the keyword STRING, there can be at most one PERC Property statement.

If *component\_subtype* is not present, then the statement applies to all instances of the *component\_type*, except for subtypes listed in a separate PERC Property statement.

## Examples

### Example 1

```
// Read width and length properties for all MP devices  
PERC PROPERTY mp w l
```

### Example 2

```
// Read width and length properties for all MN devices,  
// except for MN(na) devices, which need the area properties  
PERC PROPERTY mn w l  
PERC PROPERTY mn(na) as ad
```

### Example 3

```
// Read two string properties for all resistors  
PERC PROPERTY STRING r str1 str2
```

# PERC Report

Specification statement

## PERC REPORT *filename*

Used only in Calibre PERC applications.

### Parameters

- *filename*

A required filename of the PERC report.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

### Description

Specifies the PERC report file name. This statement must be specified once in the rule file when using PERC for rule checking.

See “[PERC Report Format](#)” in the *Calibre PERC User’s Manual* for details on the PERC report.

For information about viewing PERC results in RVE, see “[Using Calibre RVE for PERC](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [PERC Load](#), [PERC Report Maximum](#), [PERC Report Option](#), and [PERC Report Placement List Maximum](#).

### Example

```
PERC REPORT "perc.rep"
PERC LOAD perc.func INIT init_1 SELECT rule.1 rule.2
```

# PERC Report Maximum

Specification statement

**PERC REPORT MAXIMUM { *number* | ALL }**

Used only in Calibre PERC applications.

## Parameters

- ***number***

A positive integer that specifies the number of results to report for the PERC run. If this statement is not specified, the default is taken from the [LVS Report Maximum](#) statement in the rule file. The default for this latter statement is 50.

- **ALL**

Keyword that specifies all generated results for the run are reported. By default, ALL means 10000.

## Description

Specifies the number of results that appear in various sections of the [PERC Report](#). This statement may be specified once. If this statement is explicitly specified, then it overrides the LVS Report Maximum setting in a PERC run.

The upper limit of the number of internally-generated results is 10000 by default. This means that the maximum number of reportable results is also 10000 by default. This limit can be adjusted using the `perc::set_parameters -resultMaxCount` option.

See “[PERC Report Format](#)” in the *Calibre PERC User’s Manual* for details on the PERC report.

See also [PERC Load](#).

## Example

In this example, the PERC Report Maximum statement overrides the LVS Report Maximum setting.

```
LVS REPORT MAXIMUM 50
PERC REPORT "perc.rep"
PERC REPORT MAXIMUM 25
PERC LOAD perc.func INIT init_1 SELECT rule.1 rule.2
```

# PERC Report Option

Specification statement

## PERC REPORT OPTION *option* [*option* ...]

Used only in Calibre PERC applications.

### Parameters

- *option*

A required keyword that specifies the verbosity for a particular section of the [PERC Report](#).

You may specify *option* any number of times in one statement, separated by spaces. The keyword names are case-insensitive. The list of keyword names is given in [Table 4-39](#).

**Table 4-39. Keywords for PERC Report Option**

Option	Description
ALL_NET_TYPE	Reports all net types or type sets carried by a net. The default behavior is to report only the net types or type sets specified in a given rule check.
NO_DEVICE_PIN	Causes rule check results to report no pin information for device and cell instances.
NO_NET_TYPE	Causes rule check results not to report net types or type sets carried by nets. The default is to report these items. This keyword has priority over ALL_NET_TYPE if both are specified.
SKIP_PASSED_CELL	Reports only failed cells from a hierarchical run in the detailed section of the report. By default, all hcells are reported. If this option is specified, the heading of the detailed report section for failed cells changes from the standard CELL VERIFICATION RESULTS to CELL VERIFICATION RESULTS ( FAILED CELLS ONLY ).
SKIP_WAIVED_RESULT	Causes the RULECHECK WAIVED RESULTS section not to appear in the PERC Report for a run that uses waivers.

### Description

Specifies the level of certain details shown in the PERC Report. This statement may be specified any number of times. Options are accumulated across all statements.

See “[PERC Report Format](#)” in the *Calibre PERC User’s Manual* for details on the PERC report.

See also [PERC Load](#), [PERC Report Maximum](#), and [PERC Report Placement List Maximum](#).

### Example

```
PERC LOAD perc.func INIT init_1 SELECT rule.1 rule.2
PERC REPORT "perc.rep"
PERC REPORT OPTION SKIP_PASSED_CELL // only report the bad cells
```

# PERC Report Placement List Maximum

Specification statement

**PERC REPORT PLACEMENT LIST MAXIMUM { *number* | ALL }**

Used only in Calibre PERC applications.

## Parameters

- ***number***

A positive integer that specifies the maximum number of elements that appear in placement lists of the [PERC Report](#). The default is 5.

- **ALL**

Keyword that specifies all elements appear in placement lists in the report file.

## Description

Specifies the upper limit of the number of elements that appear in placement lists in the PERC Report. This statement may be specified once.

See “[PERC Report Format](#)” in the *Calibre PERC User’s Manual* for details on the PERC report.

See also [PERC Load](#) and [PERC Report Option](#).

## Example

This example sets the maximum number of items in a placement list at 25 for the specified rule checks.

```
PERC REPORT "perc.rep"
PERC REPORT PLACEMENT LIST MAXIMUM 25
PERC LOAD perc.func INIT init_1 SELECT rule.1 rule.2
```

## PERC Waiver Path

Specification statement

### PERC WAIVER PATH *filename*

Used only in Calibre PERC applications.

#### Parameters

- *filename*

A required pathname of a waiver description file.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

#### Description

Specifies that the PERC run should use waivers. This statement may be specified only once in the rule file. A Calibre Waiver license is required to use waivers. See the [Calibre Administrator’s Guide](#) for licensing information.

The waiver description file contains statements describing which results to waive during the run. A result is waived if it meets the conditions of at least one waiver statement. The waiver description file can be generated through RVE or written manually using a text editor. See “[Waiver Description File Format](#)” in the *Calibre PERC User’s Manual* for details about the file specified by the *filename* argument.

PERC waivers are discussed in detail under “[PERC Waiver Flow](#)” in the *Calibre PERC User’s Manual*. The format of the waiver data in the PERC Report is discussed under “[PERC Reporting of Waived Results](#)” in the same manual.

#### Example

```
PERC REPORT "perc.rep"
PERC LOAD perc.func INIT init_1 SELECT rule.1 rule.2
PERC WAIVER PATH "PERC.waivers"
```

# Perimeter

Layer operation

## PERIMETER *layer constraint*

### Parameters

- *layer*

A required original layer or layer set, or a derived polygon layer.

- *constraint*

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

The constraint must contain non-negative real numbers. It is interpreted in user units.

### Description

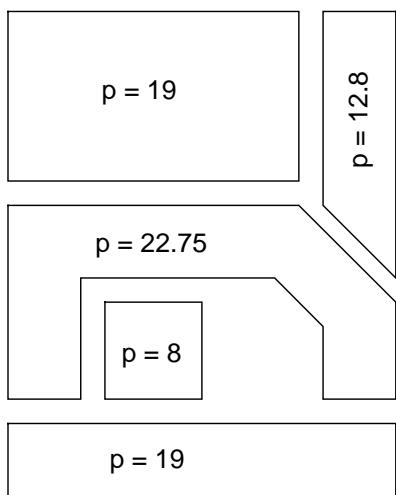
Selects all *layer* polygons with perimeters conforming to the constraint.

In the event that polygons with edges not orthogonal to the database axes exist on *layer*, you should be careful when specifying constraints in the Perimeter operation. A potential problem arises when you make an assumption about the exact size of *layer* polygons. For example, avoid using constraints such as  $= 5$  when *layer* contains non-orthogonal polygons. A better choice of constraint is  $> 4.995 < 5.005$ . This allows a tolerance of ten database units, if your [Precision](#) is 1000.

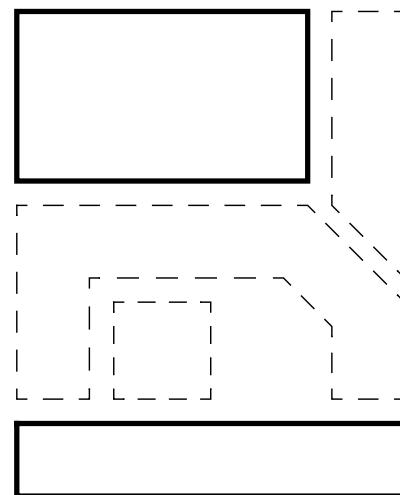
### Examples

The Perimeter operation shown in Figure 4-245 selects all layer1 polygons that have a perimeter greater than 14 but less than or equal to 20 user units.

**Figure 4-245. Perimeter**



layer1 polygons



perimeter layer1  $> 14 <= 20$

# PEX 3DReference

Parasitic extraction

**PEX 3DReference** *option\_name value [option\_name value...]*

Used in Calibre xACT 3D Reference.

## Parameters

- *option\_name value*

A required keyword pair that specifies an option name and a corresponding numerical value. This parameter must be specified at least once. All six options may be specified. Duplicate option names are not allowed. Valid options are:

<i>option_name</i>	<i>value</i>
GRID_SIZE	Must be integer value > 0.
ITERATION_LIMIT	Must be integer value > 0.
ITERATION_TOLERANCE	Must be floating point value > 0.
REGRID_ABORT_FACTOR	Must be floating point value > 0.
REGRID_LIMIT	Must be integer value >= 0.
REGRID_MULTIPLIER	Must be floating point value >= 1.

## Description

Specifies the optional parameters to be used by the capacitance solver. You must specify at least one parameter when using this statement. The fieldsolver applies default settings for options that are not specified. This statement is optional and can only be specified once.

Following is a description of the allowable option names:

**GRID\_SIZE** — Specifies the equivalent number of grid points used to discretize the problem in the Finite Difference Method (FDM). The actual number of points may be slightly more or less than what is specified. If not specified, the tool automatically detects an appropriate GRID\_SIZE for the problem.

**ITERATION\_LIMIT** — Specifies the maximum number of iterations the linear solver will go through if the iteration tolerance is not satisfied. The default value is 5000.

**ITERATION\_TOLERANCE** — Specifies the stopping criteria for the iterative linear system solver. Under normal conditions, there is no need to adjust this option. However, you may need to decrease this value when a problem involves both very large numbers and very small numbers. The default value is 1.0e–6 for capacitance.

**REGRID\_LIMIT** — Specifies the number of times the discretization should be re-drawn (regridding). Each time the discretization is re-drawn, previous discretization and corresponding charge distribution are analyzed, and a new discretization is performed. The default value is 0.

**REGRID\_ABORT\_FACTOR** — Specifies when the regrid process stops before reaching REGRID\_LIMIT. If the difference of total positive charge between two consecutive regrid runs is less than this value, then regrid will stop. For example, assume REGRID\_LIMIT=5, and the total charge in the second and third regrid runs are 1.0e-17 and 1.01e-17, respectively, or a difference of 1%. If REGRID\_ABORT\_FACTOR=0.1, then the regrid process will stop after the third regrid. The default value is 1.0.

**REGRID\_MULTIPLIER** — Specifies the multiplying factor applied to increase the number of grid points each time the discretization is redrawn. The default value is 1.

## Examples

### Example 1

To set the GRID\_SIZE to 100000 grid points and limit the number of iterations the linear solver performs to 500, use the following statement:

```
PEX 3DREFERENCE GRID_SIZE 100000 ITERATION_LIMIT 500
```

# PEX Alias

Parasitic extraction

**Note**



As of the 2010.4 release, this statement has been deprecated. Use [PEX Map](#) instead.

**PEX ALIAS** *primary\_layer* *sublayer* ... [FILL] *sublayer* ...  
[IGNORE PRIMARY]

## Parameters

- ***primary\_layer***

A required original layer or a derived polygon layer. This layer must have calibrated capacitance and resistance rules, and connectivity established. Layers may not be vias.

- ***sublayer***

One or more required original or derived polygon layers. The ***sublayer*** inherits in-die variation information, nominal electrical settings, and parasitic calculations from ***primary\_layer***.

Sublayers should all map to the same physical layer. There must be no geometric overlap in the layout between ***sublayers***. Overlapped sections are not detected and cause inaccurate results.

- **FILL**

An optional keyword set identifying the sublayers following it as metal fill. Identifying metal fill results in faster, smaller runs in some situations.

- **IGNORE PRIMARY**

An optional keyword specifying that ***primary\_layer*** should only be used for property and rule inheritance. All geometric data on this layer will be ignored. This can be used when the primary layer overlaps all or part of the secondary layers.

## Description

Specifies layers that use the capacitance, resistance and in-die variation rules of ***primary\_layer*** unless otherwise specified. The ***primary\_layer*** represents the physical layer as described in the calibrated rule file. The ***sublayers*** represent the derived layers that result from the operations necessary to extract the designed devices.

If the ***primary\_layer*** is also listed as a ***sublayer***, then any PEX IGNORE commands on the ***primary\_layer*** will not be inherited by the other listed ***sublayers***. See Example 3 under the Examples section for this statement.

Any layers named in ***primary\_layer*** or ***sublayer*** may not appear in any other PEX Alias specification. Layers that appear in the same PEX Alias statement and also the [Capacitance](#)

**Order** statement must be in the same sequence. (The sublayers are not required to be in the Capacitance Order statement.)

**Note**



Connections are not inherited. All **sublayer** connections must be explicitly specified in **Connect** and **Connect By** statements.

Rules are not inherited in the following conditions, by order of precedence:

- The **sublayer** does not appear in a Connect statement. The layer is treated as nonconducting and does not have electrical properties.
- The rule defining the interaction of the layer(s) is overridden with **PEX Ignore Capacitance** or **PEX Ignore Resistance**. The effect specified in PEX Ignore is not calculated. Other rules are inherited as normal.
- A **sublayer** has its own properties explicitly defined. The explicitly defined properties and rules are used instead of the inherited ones. A **sublayer** may explicitly define a base property such as resistance and still inherit in-die effects from **primary\_layer**.
- Rules for **primary\_layer** are overridden with a PEX Ignore statement. This causes the specified effect to not be calculated for both **primary\_layer** and all its **sublayers**. To ignore effects on only the **primary\_layer**, use the IGNORE PRIMARY keyword set in the PEX Alias statement.

The PEX Alias statement can be used with **PEX Ignore Capacitance** and **PEX Ignore Resistance**. An error results if the rule file contains **PEX Elayer**.

If the parasitic extraction requires density calculations, then density values for the layers in the same PEX Alias statement are added, and the resulting total density used for any one of the layers.

## Examples

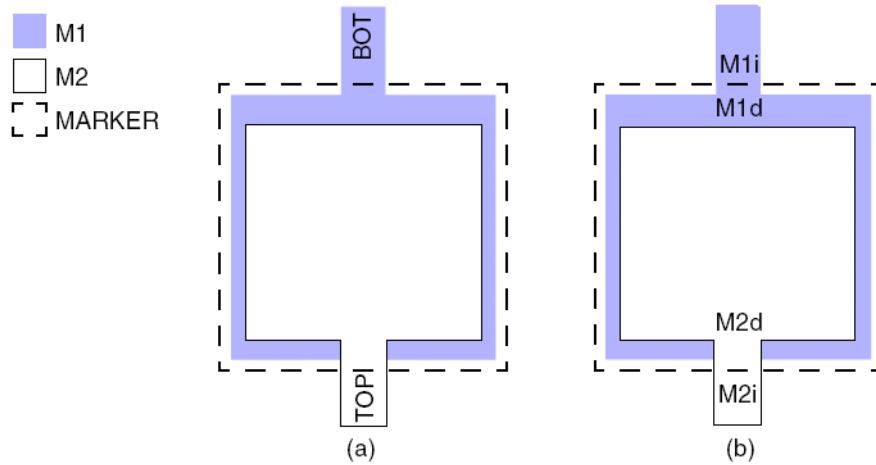
### Example 1

This example shows how the PEX Alias statement can be applied to a capacitor device.

The layout in [Figure 4-246 \(a\)](#) shows a device parallel plate capacitor made of M1 and M2 layers. TOP and BOT are the device terminals. The device is indicated by everything that is enclosed by the MARKER polygon. [Figure 4-246 \(b\)](#) shows how Boolean operations between M1, M2, and MARKER layers are used to generate new device and interconnect layers.

The following layer operations generate device and interconnect layers shown in [Figure 4-246 \(b\)](#).

```
// Layers that define the capacitor device, or body
M1d = M1 and MARKER
M2d = M2 and MARKER
// Layers that define the interconnect to the capacitor
M1i = M1 not MARKER
M2i = M2 not MARKER
```

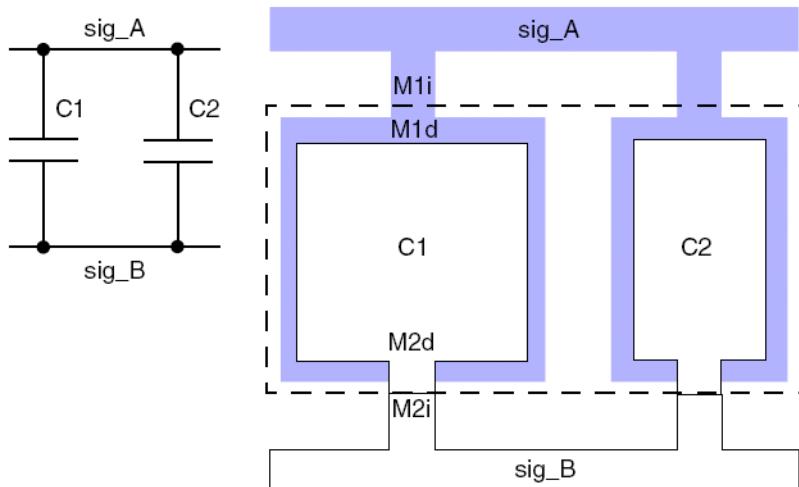
**Figure 4-246. Metal1-Metal2 Capacitor Layout**

You can assign an alias to the device and interconnect layers as follows:

```
// Layer Aliasing
PEX ALIAS M1 M1d M1i
PEX ALIAS M2 M2d M2i
```

The first statement assigns the properties of M1, the primary layer, to the generated layers M1d and M1i. The second statement does likewise for the layers generated based on M2.

[Figure 4-247](#) shows an example of a simple circuit composed of two M1-M2 parallel plate capacitors. During LVS, the [LVS Device Type](#) statement calculates parameters such as area and perimeter for C1 and C2. Since the rule statements use PEX Alias, layers M1d, M2d, M1i, and M2i are used for these calculations instead of the original layers, M1 and M2.

**Figure 4-247. Two Parallel M1-M2 Capacitors**

You can extract the parasitic capacitors between C1 and C2 without extracting the device capacitors. Do this by using PEX Alias combined with [PEX Ignore Capacitance](#), as follows:

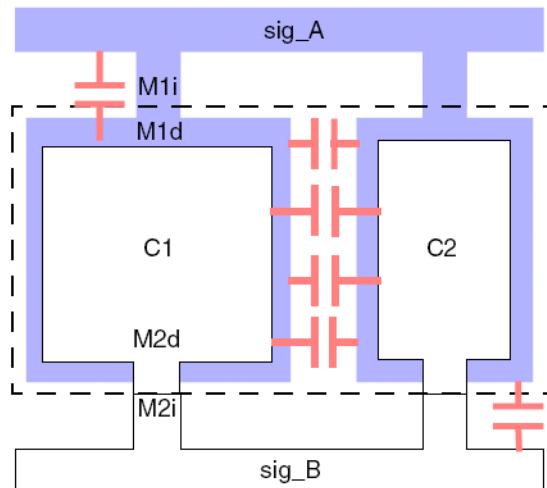
```
// Generate derived layers
M1d = M1 and MARKER
M1i = M1 not MARKER
M2d = M2 and MARKER
M2i = M2 not MARKER

// Alias definitions
PEX ALIAS M1 M1d M1i
PEX ALIAS M2 M2d M2i

// Ignore capacitance between device bodies
PEX IGNORE CAPACITANCE ALL M1d M2d
```

The last statement indicates that all capacitances between layers M1d and M2d in a device will be ignored. The capacitance between these layers will be part of the device model. However, the extracted netlist will include capacitances between M1d, M1i, M2d, and M2i between C1 and C2. [Figure 4-248](#) shows some of the capacitors that may be extracted.

**Figure 4-248. Parasitic Interaction Between M1-M2 Capacitors**



Parasitic capacitors between layers M1d and M2d of C1 and C2 will be extracted as well as between M1i and M2d, and M2i and M1d.

#### Example 2

This example shows how the PEX Alias statement is used with PEX Ignore Resistance statement. In this case resistance will be ignored for all layers listed as aliases, even though only the A layer is specified by the PEX Ignore Resistance statement:

```
// Alias definition
PEX ALIAS A B C

// Ignore resistance for all layers A, B, and C
PEX IGNORE RESISTANCE A
```

### Example 3

This example shows how apply the PEX Ignore Resistance effect only to the layer specified in the PEX Ignore Resistance statement:

```
// Alias definition  
PEX ALIAS A A B C
```

If the primary layer A is also specified as an aliased layer, then PEX Ignore ... statements on the primary layer will not automatically be inherited by layers B and C. This means that the following statement only ignores resistance for layer A:

```
PEX IGNORE RESISTANCE A
```

To ignore resistance for layers B and C, explicitly define ignore statements for each layer:

```
PEX IGNORE RESISTANCE B  
PEX IGNORE RESISTANCE C
```

# PEX BA Mapfile

Parasitic extraction

## PEX BA MAPFILE *mapfilename* [CALIBREVIEW]

### Parameters

- *mapfilename*

A required back-annotation mapfile name. The mapfile contents are described in the sub-section “[Mapfile Contents](#)”.

- CALIBREVIEW

An optional keyword that instructs the formatter to suppress the automatically generated LPE values from the Calibreview netlist output. With this option, only LPE values specified in the mapfile are written to the netlist.

### Description

Specifies a file containing the device, pin name, and model mappings used to map the extracted intentional devices in a parasitic netlist to their corresponding simulation models. This is used in backannotation postlayout analysis. The mapfile has this structure:

```
//version: 2

%Devices
SourceDev NumRepChar Ins/RepString OptNetlistModel //OptComments
...
%Pins
SourceDev SourcePin RepString OptSrcPin OptRepString //OptComments
...
%Models
ReqOrigM ReqRepM OptIns/Rep OptPinChange OptParmChange OptLPE //OptComment
...
```

where each section may have multiple lines, and “Opt” indicates an optional entry. The sections can appear in any order, but the models section is processed last. Use the model mappings for most changes, and use the pins and devices mappings for device-specific transformations. The commented version line must be the first line in the file and must be a value of 2 with a space between the colon and 2. Any value other than 2 given for version will imply that the mapfile format is the old format and OptLPE syntax will be ignored. The CalibreView netlist format output contains the LPE parameters by default. Use the CALIBREVIEW option to override this default and only include the LPE parameters specified in the mapfile in the generated netlist.

---

**Note**



Intentional devices in CalibreView always need model names. When generating CalibreView output, if the mapfile deletes the model name or replaces it with a blank name the formatter uses the original model name.

## Mapfile Contents

The mapfile consists of three sections: devices, pins, and models. The rules in the sections are executed from the beginning of the section to the end, so the first match in each section is applied. Order the rules with the most specific rules at the beginning of the file and the most general rules at the end. You can insert comments in the mapfile using a double-slash (//). The first line in the mapfile must be the comment //version: 2, with a space provided between the colon and 2. The value given for version must be 2. Any other value given for version will imply that the mapfile format is an old format and any LPE syntax used will be ignored.

**Devices** — The device map section begins with the required keyword “%Devices”. Entries have this syntax:

*SourceDev NumRepChar Ins/RepString [OptNetlistModel] [//OptComments]*

- **SourceDev** — Specifies the device to be backannotated. All the devices with matching names have the same rule applied to them. You can use an asterisk character (\*) for device name wildcards.
- **NumRepChar** — Specifies the number of characters to replace. To insert characters, set the number to 0. Characters are inserted or replaced from the first letter of the SourceDev. See [Examples](#).
- **Ins/RepString** — Specifies the new characters to insert or use to replace the original name.
- **OptNetlistModel** — Specifies the optional netlist model. This column can be empty if you do not want to replace the model. You can use two double-quotes (“”) to delete the model name entirely.

**Pins** — The pin map section begins with the required keyword “%Pins”. The pin map section only has replace mode. Entries have this syntax:

*SourceDev SourcePin RepString [OptSrcPin] [OptRepString] [//OptComments]*

- **SourceDev** — Specifies the device to be backannotated. You can use an asterisk to match more than one device at a time.
- **SourcePin** — Specifies the pin to be backannotated.
- **RepString** — Specifies the new string to replace the original name.
- **OptSrcPin** — An optional entry for a subsequent pin to be backannotated.
- **OptRepString** — An optional entry that specifies the new string to replace the original name.

**Models** — The model map section begins with the required keyword “%Models”. Entries have this syntax:

*ReqOrigM ReqRepM [OptIns/Rep] [OptPinChange] [OptParmChange] [OptLPE...] [//OptComment]*

- **ReqOrigM** — Specifies the original model name to which one of the following BA mapfile enhancements are applied:
    - Replace this model name with a new name.
    - Change the pin names.
    - Output LPE values to the netlist.
- Use netlist model names for layout-based extraction. Use source device model, subtype, or device element name [subtype model name] for source-based extraction. When using device element name [subtype model name], you must enclose the subtype model name in either parentheses ( ), or square brackets ( [ ] ). You can use an asterisk (\*) to match more than one model at a time.
- **ReqRepM** — Specifies the replacement model name. Use “” to delete the model name. This entry is required; if you do not want to replace the model name, use the same name as in ReqOrigM.
  - **OptIns/Rep** — An optional section for modifying the instance name. The section consists of the keyword INST followed by a number of characters to replace at the beginning of the name, and the text to replace those characters with. Set the number of characters to 0 for inserting a prefix; for example, INST 0 pre\_ to insert “pre\_” before all instance names.
  - **OptPinChange** — An optional section for renaming or remapping pins. The section consists of the keyword PIN (for renaming) or PINORDER (for remapping) and then the change pairs, original pin name then new pin name. There can be any number of changes in the section but both the original and the new name must be specified.

PIN and PINORDER are mutually exclusive. Use PIN for replacing pin names; use PINORDER for mapping device pin names and placing the pins in the specified order in the parasitic netlist.

When using PINORDER, all required pins for a device must be specified or the formatter will issue a warning. List the pins in the desired order with new names. A device’s required pins are defined in the Device statement. The default order is that of the SPICE model. For example, a MOS transistor’s pins are listed in the Device statement in the order G, S, D, B; and B is optional. The SPICE model’s order is D, G, S, B. To swap the S and D pins, the PINORDER section would look as follows:

```
PINORDER S new_s_name G new_g_name D new_d_name
```

Notice that B, being an optional pin, does not need to appear unless it is being remapped. The original pin name (G, D, and S in the example) should match the pin names in the device statement.

To add a new pin to a device include the @ADD keyword subsection for PINORDER:

```
PINORDER [old_pname new_pname]... [@ADD old_pname added_pname position]
```

where

- **PINORDER** is a required keyword indicating the beginning of the optional section that remaps the pins on the device.
- *old new* are the old pin name and the new pin name. The number of pairs of pins should map to the existing pins for the device.
- **@ADD** *old\_pname added\_pname position* is an optional section that takes an existing pin and copies it to create a new pin in a specified position. The old pin name, *old\_pname*, represents the pin that will be duplicated. The added pin, *added\_pname*, represents an existing pin on the device that may not be netlisted. The *position* represents the position in the list of pins. If the position specified is greater than the pin count, then pin count is increased by 1.

For example,

```
PINORDER S snew G gnew D dnew @ADD G Bnew 4
```

---

 **Note**

The rules in the Devices and Pins sections are applied before the rules in the Models section. This may transform the expected model name or pin name and cause a mismatch.

---

- **OptParmChange** — An optional section for modifying parameter names and adding or deleting parameters. This section has the syntax:

**PARM** [*old new*]... [**@ADD** *parm=value* [*parm=value*]...] [**@DEL** *parm...*]

where

- **PARM** is a required keyword indicating the beginning of the optional section.
- *old new* are the old parameter name and the new one. Any number of pairs can be specified.
- **@ADD** *parm=value* is an optional section that adds parameters to the device definition. Any number of parameters can be added.
- **@DEL** *parm* is an optional section that deletes parameters from the device definition. If *parm* does not match any device parameters it is ignored.

There must be at least one change (renaming, adding, or deleting) in the section. Adding and deleting parameters occurs before renaming. For example,

```
PARM weff W leff L @DEL L W
```

will delete parameters L and W, and then rename leff to L and weff to W.

- **OptLPE** — An optional section for adding layout parasitic extraction (LPE) parameters to device instance statements for specific intentional device models in a parasitic netlist.

**Note**

 The CalibreView netlist format output contains the LPE parameters by default. To only include the LPE parameters specified in the mapfile use the CALIBREVIEW option.

One or more LPE sections can be defined for a given model. An LPE section has the syntax:

**LPE** [*lpe\_name*] [**NORC** *value*] [**C** *value*] [**R** *value*] [**RC** *value*]

where

- **LPE** is a required keyword indicating the beginning of the optional section.
- *lpe\_name* is an optional name which can be specified for the LPE and will appear in the output netlist in place of the LPE keyword.
- **NORC** *value* is an optional keyword value pair used to specify that a net that has no R or C associated with it, otherwise referred to as an ideal net.
- **C** *value* is an optional keyword value pair used to specify that lumped or coupled capacitance should be controlled.
- **R** *value* is an optional keyword value pair used to specify that distributed resistance should be controlled.
- **RC** *value* is an optional keyword value pair used to specify that distributed resistance and capacitance or distributed resistance and coupled capacitance should be controlled.
- *value* specified for C, R, RC, or NORC must be an integer and is derived from the device model simulation.

There must be at least one type of parasitic (C, R, RC, or NORC) specified in a given LPE section. For example,

```
cmim5_2p cmim5_2p INST 1 X PINORDER pos minus neg plus LPE NORC 0
```

There are two ways to control which LPE parameters are written to the netlist:

- Use the [PEX Netlist LPE Using Extmode](#) statement to control the addition of LPE parameters using the extraction mode specified in the PDB stage.

- For incremental or selected net extraction, device terminals can be connected to nets with differing parasitic models. In that case, the device shall be assigned a worst case LPE value based on the following table:

**Table 4-40. “Worst Case” Parasitic Values for LPE**

Parasitic Model A	Parasitic Model B	Used for LPE Setting
No parasitics	No parasitics	No parasitics
	C	
	R	
	RC	
C	C	C
	R	No parasitics
	RC	C
R	R	R
	RC	
RC	RC	RC

## Examples

### Example 1

This is a MAPFILE example. Explanations are included in the comments section of each line.

```
//version: 2

%Devices
MNO    0X_    mp_model //for all MNO devices, insert X_; MNO -> X_MN0
C4     1  X   c_model  //replace 1st char C with X; C4 -> X4
R3     1  B   r_model  //replace 1st char R in R3 with B; R3 -> B3
R*    1  X      //replace all Rs with X (except for R3); R2 -> X2
MP_*  2  X   modsf    //replace 2 chars MP with X; MP_I$4 -> X_I$4

%Models
// The example below acts on the model NS to replace the first
// character of the instance name with "X_", rename the neg and pos pins
// to minus and plus, and add an LPE value of 1, 2, or 3 depending
// on extraction type.
NS  NS  INST 1 X_    PIN neg minus pos plus  LPE C 1 R 2 RC 3

%Pins
C4 neg  minus p2  plus2      //replace C4:neg with C4:minus and
                                // C4:p2 with C4:plus2
R2 pos  plus      neg  minu2 //replace R2:pos with R2:plus and
                                // R2:neg with R2:minu2
R* neg  minus      //for all other R, replace neg with minus
M* d    drain      //for all MOS devices, replace d with drain
```

**Example 2**

This example shows how the models section differs between layout-based and source-based extraction. The layout-based flow uses the netlist model name; this is specified in the Device statement. The source-based flow uses the LVS model name.

```
////////// This set is for layout based flow
%Models
RPO1SAM1 "" INST 1 X PINORDER neg jukn pos ksfplus PARM w WTT \\1
enllgp_bs3ju enllgp_bs3ju INST 0 X_ parm L length \\2
enllgn_bs3ju foo2 INST 0 X_ parm l length WADF W \\3

////////// This set is for source based flow
ns      ""      INST 1 X PINORDER neg jukn pos ksfplus PARM w WTT
p       p      INST 0 X_ parm L length \\2
N      foo2     INST 0 X_ parm l length WADF W \\3
```

These two cases do the same thing; only the model names differ. The first mapping deletes the model name; changes the first character of the device name to X; remaps and renames the pins; and in the parameters, replaces w with WTT in the output. The second and third mappings in both sections replaces the model name, puts an X\_ before the device name, and replaces parameters.

**Example 3**

This is a portion of a MAPFILE containing various LPE usage examples.

```
//version: 2

%Models
// The example below acts on the model NS to replace the first
// character of the instance name with "X_", rename the neg and pos pins
// to minus and plus, and add an LPE value of 1, 2, or 3 depending
// on extraction type.
rhiporpo rhiporpo INST 1 X PINORDER pos plus neg minus sub b LPE NORC 0 C
100 R 200 RC 300 LPE foo2 NORC 0 C 100 R 200 RC 300

cmim5_sh cmim5_sh INST 1 X PINORDER pos minus neg plus sub sh LPE C 1 R 2
RC 3

cmim5_2p cmim5_2p INST 1 X PINORDER pos minus neg plus //no LPE given

cmim5 cmim5 INST 1 X PINORDER pos minus neg plus sub psub LPE foo3 NORC 0
C 10 R 22 RC 38
```

**Example 4**

This example shows how the models section uses device element names together with device model names for source-based extraction.

```
////////// This set is for source based flow
M[ND] nch3
R[ND] rnodrpo
M[PD] pch3
R[PD] rpodrpo
```

Where M and R represent the device element names, and ND and PD represent the device model names. The replacement model names are nch3, rmodrpo, pch3 and rpodrpo.

### Example 5

To use the CALIBREVIEW keyword to control the output of LPE values do the following.

- Add the following statements to your rule file:

```
PEX NETLIST mynetlistfile CALIBREVIEW
PEX BA MAPFILE mymapfile CALIBREVIEW
```

- Add LPE specifications to your mapfile. For example:

```
%Models
N N LPE foo NORC 0 C 100 R 200 RC 300
P P LPE NORC 0 C 1 R 2 RC 3
```

- Run extraction for your design.
- Create the cell mapfile, adding the corresponding LPE information from the mapfile:

```
...
(n
  ( mgc_n4 nmos4 symbol)
  ( (b B) (d D) (g G) (s S) )
  (1 1) (w w) (m m)
    (w wscale1 (w*7))
    (w wscale2 (w*15))
    (lpe flag1 (if foo=0 || foo=200 then flag1=1 else flag1=0))
    (lpe flag2 (if foo=0 || foo=100 then flag2=1 else flag2=0))
  )
)

(p
  ( mgc_p4 pmos4 symbol)
  ( (b B) (d D) (g G) (s S) )
  (1 1) (w w) (m m)
    (w wscale1 (w*17))
    (w wscale2 (w*125))
    (lpe lpe_a (if lpe==0 || lpe==1 then lpe_a=1 else lpe_a=0))
  )
)
...
```

- Load the extracted netlist into Cadence Virtuoso.

See the “[Syntax for Cellmap Statements](#)” section of the *Calibre Interactive and Calibre RVE User’s Manual* for more information and additional examples.

# PEX Bulk Layer

Parasitic extraction

**PEX BULK LAYER** *layername* [*layername* ...]

---

**Note**

 As of the 2008.3 release, this statement has been deprecated. Use [PEX Resistance Parameters](#) **BULKRESISTANCE** instead.

---

## Parameters

- *layername*

A required layer name identifying substrate or well layer. The specified layers must have connectivity. Multiple layers may be identified. A layer must be non-resistive; that is, no sheet resistance (RSH) or resistivity (RHO) values should be defined for it.

---

**Note**

 If a layer is used that has a defined RSH or RHO value, then the extraction run stops with the following error message:

Resistive layer <NAME> is specified in PEX BULK LAYER statement

---

## Description

Specifies the layer or layers used for connecting to the bulk pins of transistors. The bulk node of a transistor is normally shorted to a power or ground potential. However, to increase the accuracy of IR drop modeling to a transistor's bulk pin, this statement can be used to connect the bulk pin to the nearest substrate or well tap. Substrate resistance is not extracted with this statement.

The closest distance to a substrate or well tap is defined as the shortest span between the middle of the device and the middle of the nearest tap, within the same bulk layer polygon. Taps in a different bulk layer polygon are not considered as connected to the bulk.

Flat extraction with DSPF and HSPICE output is supported. The supported extraction types are -r, -rc, and -rcclm. The -cc mode is not supported. This statement should be used with a block that has 100 to 1000 transistors.

Transistors with a bulk connection must have a pin named "b" or "bulk." The bulk pin must be connected to the same net as the nearest bulk tap. For example, if the bulk pin of an NMOS transistor is connected to net VSS:1, then the p-substrate tap must also be connected to VSS:1 to be considered as the nearest connection.

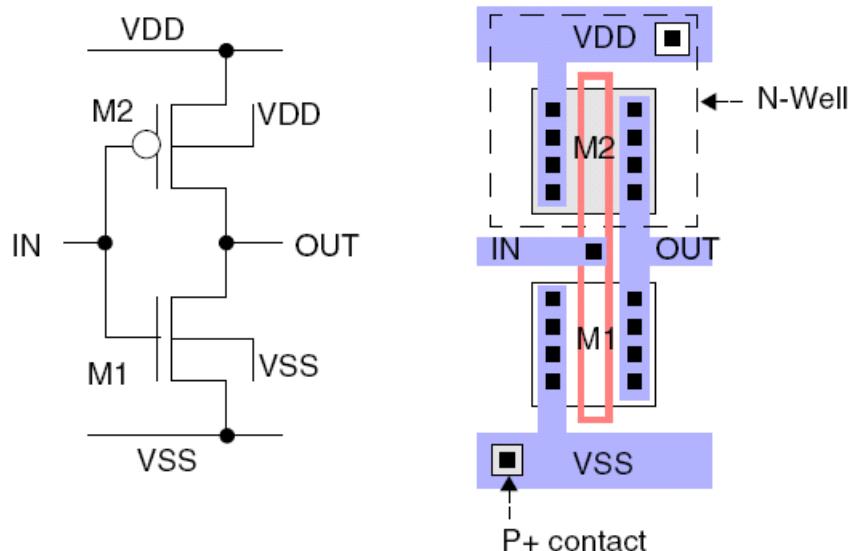
## Examples

The following example shows how this statement might be used. [Figure 4-249](#) shows an inverter schematic and its layout. The HSPICE netlist for this inverter is

```
* Inverter
* Pin Order: Mxx Drain Gate Source [Bulk] Length Width Model
XM1 VSS IN OUT VSS L=1 W=5 NMOS
XM2 VDD IN OUT VDD L=1 W=5 PMOS
```

The pplus contact on VDD and nplus contact on VSS shown are typical of a cell-level layout and would be considered as close taps to the n-well and p-substrate, respectively.

**Figure 4-249. Simple Inverter Layout With Ideal Bulk and N-Well Taps**



A cross section of M1, an NMOS transistor, is shown in [Figure 4-250](#). In this case, it would not be necessary to account for an IR drop between the bulk pin of the NMOS and the p-substrate contact.

**Figure 4-250. Cross Section of Ideal PSUB Tap for NMOS**

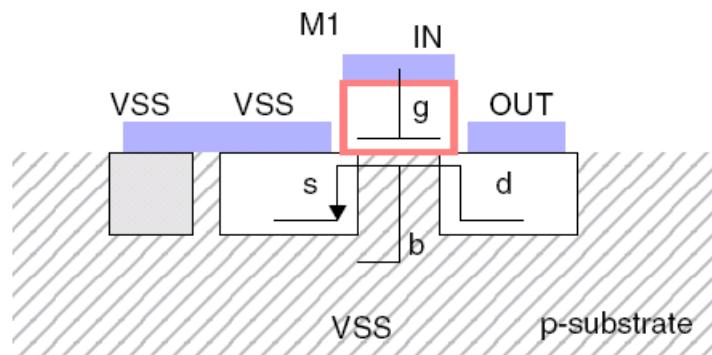


Figure 4-251 shows a case where the nearest substrate connection is found in a nearby cell. In this case, finding the nearest substrate contact may be necessary.

**Figure 4-251. Non-Ideal PSUB Tap for NMOS**

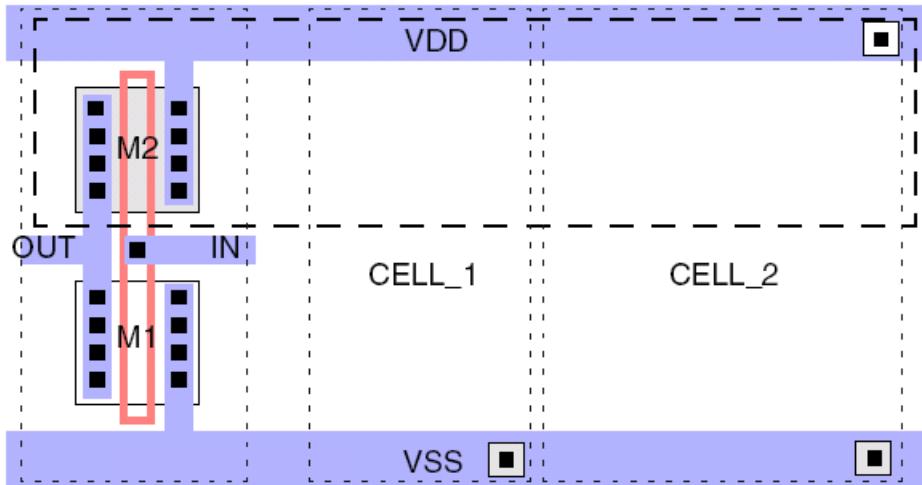
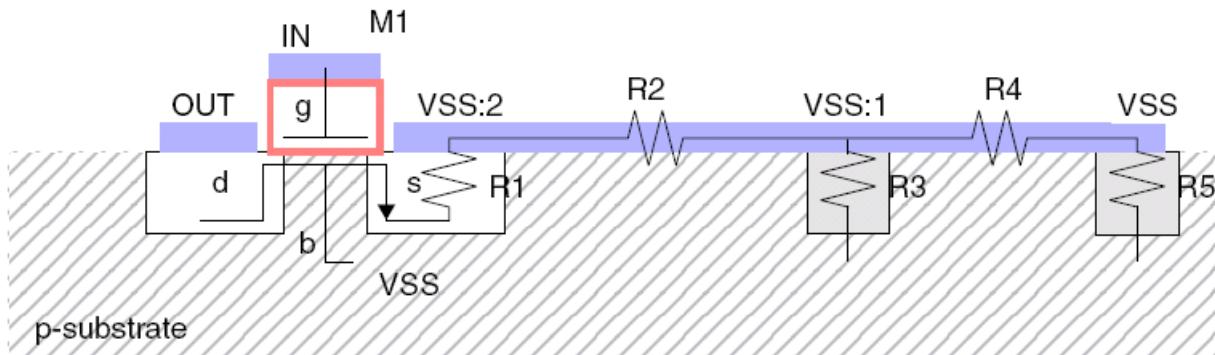


Figure 4-252 shows the extracted circuit in the immediate vicinity of M1 if PEX Bulk Layer is not used. The netlist is:

**Figure 4-252. Model of Extraction Without PEX Bulk Layer**



\* Inverter

XM1 OUT IN VSS:2 VSS L=1 W=5 NMOS  
XM2 OUT IN VDD VDD L=1 W=5 PMOS

\*\* Parasitic Resistors

\*\* Pin Order: POS NEG Value

R1 VSS:2 M1:s 1

R2 VSS:2 VSS:1 2

\* R3 is considered a dangling resistor, will not appear in netlist

R4 VSS:1 VSS 2

\* R5 is considered a dangling resistor, will not appear in netlist

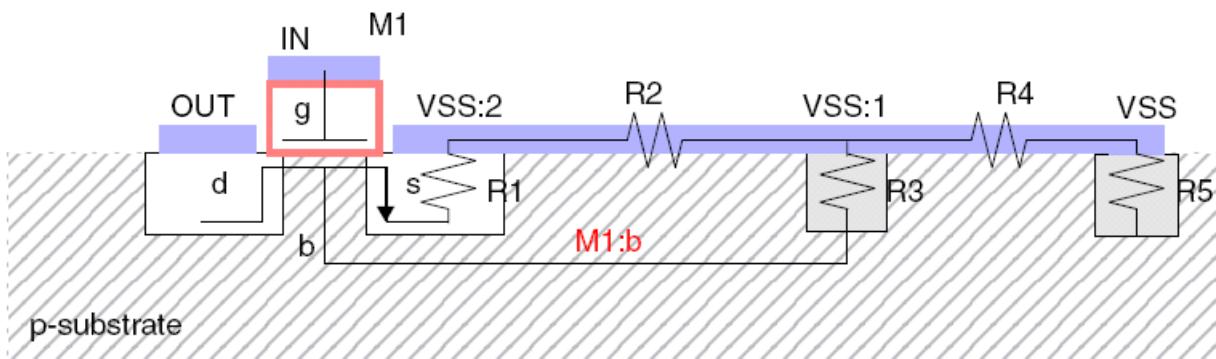
## PEX Bulk Layer

Including the PEX Bulk Layer statement as shown below will change the results of the extraction by accounting for the closest found p-substrate tap, in this case within the cell boundary of CELL\_1. The SVRF statements to do this are:

```
// explicit p-substrate layer definition  
BULK = EXTENT  
PSUB = BULK NOT NWELL  
  
// layers that bulk pins can connect to  
PEX BULK LAYER PSUB NWELL
```

Figure 4-253 shows the resulting circuit. Note that a new net, M1:b, is introduced. This intermediate net is used to connect the bulk pin of M1 to the nearest p-substrate tap.

**Figure 4-253. Model of Extraction With PEX Bulk Layer**



The resulting netlist is:

```
* Inverter  
XM1 OUT IN R1:NEG XM1:b L=1 W=5 NMOS  
XM2 OUT IN VDD XM2:b L=1 W=5 PMOS  
  
* Parasitic Resistors  
R1 VSS:2 XM1:s 1  
R2 VSS:2 VSS:1 2  
R3 VSS:1 XM1:b 1  
R4 VSS:1 VSS 2  
* R5 is considered a dangling resistor, will not appear in netlist
```

# PEX CMP Mode

Parasitic extraction

## PEX CMP MODE

{VCMP SUFFIX | PRAESAGUS FILENAME} *string*  
 [DIRECTORY *path*]  
 [DEFAULT\_DENSITY *value*]

### Parameters

- **VCMP SUFFIX**  
 Specifies using TSMC's Virtual Chemical Mechanical Polishing (VCMP) data format.
- **PRAESAGUS FILENAME**  
 Keyword set that specifies using the Praesagus Copper Prediction and Verification database.
- ***string***  
 Specifies a filename for PRAESAGUS mode or a suffix for VCMP mode.
- **DIRECTORY *path***  
 Optional keyword and value set for using either **VCMP** or **PRAESAGUS** parameters where *path* defines the directory location for the VCMP files.
- **DEFAULT\_DENSITY *value***  
 Optional keyword and value set used with feature level VCMP models only. If not defined, the default for *value* is 0.4, or 40% density.

### Description

This statement specifies the type of CMP mode used to handle layer thickness variation during parasitic extraction. Metal layer thickness variations are used for capacitance calculations and Rho-based resistance calculations.

Layer thickness is defined as a function of drawn width and drawn space and also expressed by equations that are a function of drawn width and density. The CMP data must be provided by the foundry.

### Examples

To use VCMP mode with file suffix *z2o.txt* and a density of 30%, use the following statement:

```
PEX CMP MODE VCMP SUFFIX "z2o.txt" DIRECTORY "./myInput/vcmpDir"
  DEFAULT_DENSITY 0.3
```

In this case a VCMP file for the metal1 layer would be located in:

```
./myInput/vcmpDir/metal1z2o.txt
```

To use the PRAESAGUS mode with a file, you can add one of the following statements:

```
PEX CMP MODE PRAESAGUS FILENAME "./myInput/psgs.txt"
```

## **PEX CMP Mode**

---

or

```
PEX CMP MODE PRAESAGUS FILENAME "psgs.txt" DIRECTORY "./myInput"
```

In both cases the complete file path is:

```
./myInput/psgs.txt
```

# PEX Contact Capacitance

Parasitic extraction

## PEX CONTACT CAPACITANCE {YES | NO}

### Parameters

- **YES**

A keyword that includes contact effects in capacitance calculations. This is the default.

- **NO**

Disables contact capacitance extraction. Must be specified to disable this feature.

### Description

This optional statement can be used to turn off calculations for all contact capacitance. When used with [PEX Via Capacitance](#), these statements control contact and via extraction. If neither statement appears in a rule file, then via and contact capacitances are extracted.

### Examples

To make the default behavior explicit in your rule file, add the following line:

```
PEX CONTACT CAPACITANCE YES
```

To exclude contacts from extraction, add the following statement:

```
PEX CONTACT CAPACITANCE NO
```

If both of the following statements are present in the rule file, then contact capacitance will be extracted:

```
PEX VIA CAPACITANCE NO  
PEX CONTACT CAPACITANCE YES
```

## PEX Corner

Parasitic extraction

**PEX CORNER** *name* [*name...*]

### Parameters

- *name*

A required name identifying the corners for which the capacitance and resistance rules have been calibrated. Each ***name*** must be unique and start with a letter or number. The first named corner is always “typical”. The corner names are specified during the calibration process.

### Description

The PEX Corner statement names the foundry-supplied process corners and is generated by the calibration process using the xCalibrate rule file generator.

This statement indicates the rule file contains the information for extracting multiple corners, and you do not need additional corner-specific calibrated rules.

---

#### **Caution**

---



Do not modify the PEX Corner statement.

---

All corners named in this statement are available for single-run extraction and individual netlisting. Having multiple corners defined in one file speeds up multi-corner extraction when compared to extracting each corner separately.

To add additional corners, use the [PEX Corner Custom](#) statement.

### Examples

The following statement is an example from a rules.C file with three process corners:

```
PEX CORNER typical foundry_wcs foundry_wcp
```

The corner-specific rules use these names to identify which corner they act upon.

# PEX Corner Custom

Parasitic extraction

```
PEX CORNER CUSTOM name [name...] [REFERENCE base [base...] ]
{ VARIATION layer [layer]
  parameter [PROFILE profile_name] {DELTA | PERCENT} diff [diff...]
    [parameter {DELTA | PERCENT} diff [diff...] ...]
  { VARIATION layer [layer]
    parameter [PROFILE profile_name] {DELTA | PERCENT} diff [diff...] }...
```

## Summary

Optional statement that specifies variations on foundry-supplied process corners. The rules calculating parasitic capacitance and resistance must include functions for process variation.

## Parameters

- ***name***

A required name for the corner that will be calculated. You can specify any number of additional names. Names cannot be identical to the foundry-supplied process corner names.

- **REFERENCE** *base*

An optional keyword set specifying the foundry-supplied process corner(s) from which each user-defined corner is calculated. If this keyword set is not present, all corners will be based on the typical process corner.

The parameter *base* must be specified the same number of times as ***name***. Valid foundry corner names can be identified from the **PEX Corner** statement, found in the calibrated rule file.

- **VARIATION** *layer* [*layer*] { ***parameter*** [PROFILE *profile\_name*] **DELTA** *diff* [*diff...*] }

A required keyword set specifying the variation for an interconnect layer (1 *layer* specified) or via layer (2 *layers* specified). You can specify any number of additional layer variations. Each layer variation must include at least one parameter to be changed.

***layer*** — The name of the layer to which these variations apply.

**parameter** — One of the following:

- THICKNESS
- WIDTH
- DIELECTRIC\_CONSTANT
- N1
- N2
- RSH
- RHO

---

**Note**



RHO always has precedence over RSH. This means that if the base corner supplies RHO information and PEX Corner Custom gives RSH information, the RSH variation is seemingly ignored because RHO was used.

---

**PROFILE** *profile\_name* — For DIELECTRIC\_CONSTANT, an optional keyword and *profile\_name* indicating a specific profile for the variation. If PROFILE is not specified, the variation applies to all profiles. Specifying PROFILE when **parameter** is not DIELECTRIC\_CONSTANT causes a warning.

**{DELTA | PERCENT} diff** — The adjustment to be applied to the corresponding value in the foundry corner. The parameter *diff* must be specified the same number of times as *name*.

## Description

The PEX Custom Corner statement is an optional statement that specifies variations on foundry-supplied process corners. You can use this statement to generate a custom corner for your technology based on foundry corners provided the [PEX Corner](#) statement. The PEX Corner statement is created by xCalibrate and is not user-defined.

Corners are created when extraction is run. You can specify a subset of corners to extract or write out by using the -corner switch during invocation. The netlists for corners are named *netlist\_name*, where *netlist* is the *filename* specified in the PEX Netlist statement and *name* is the name specified in the PEX Corner Custom statement.

---

**Note**

Custom corners are calculated from existing foundry corners. For example, you may start with values from the typical corner and specify metal thickness or width variations based on values close to the typical values. PEX Corner Custom provides a mechanism for managing these user-specified variations. The foundry-provided calibrated rules are the only source of in-die information.

---

The REFERENCE *base* set should be used if any corner parameters are inherited from the base corner rather than explicitly specified in the statement. If the original corner calibration included in-die variation, as indicated by the line “// #define \$XCAL\_EXCL\_INDIE” in the calibrated rules, you will need to include this keyword set in order to inherit the in-die information.

## Examples

### Example 1

The following is a minimal PEX Corner Custom statement:

```
PEX CORNER CUSTOM udc VARIATION m1 RSH DELTA 0.051
```

### Example 2

The following statement provides two corners based on a typical process corner.

```
PEX CORNER CUSTOM u1 u2 REFERENCE typical typical
VARIATION poly                                //Poly variation
thickness delta -0.0074 0.0144
width delta   -0.003 0.006

VARIATION metal1                            //Metal variations
thickness delta -0.0249 0.0495
width delta   -0.0095 0.019

VARIATION metal2
thickness delta -0.0275 0.0548
width delta   -0.0105 0.021

VARIATION metal3_diel1                      //Dielectric variation
thickness delta -0.2344 0.4388
dielectric_constant delta 0 0
```

## PEX DEF Extract Cell Obstructions

Parasitic extraction

### PEX DEF EXTRACT CELL OBSTRUCTIONS {NO | YES}

#### Parameters

- **NO**

Specifies not to include obstruction geometries during extraction. This is the default behavior.

- **YES**

Specifies to include obstruction geometries.

#### Description

Specifies that obstruction geometries are treated as layout objects and included as ground when calculating parasitic capacitance. Obstruction geometries are used in LEF files to block routing. By default, Calibre xRC does not include obstructions in extraction.

#### Examples

To include obstruction geometries during extraction, put the following statement in the rule file:

```
PEX DEF EXTRACT CELL OBSTRUCTIONS YES
```

# PEX DEF Map

Parasitic extraction

## PEX DEF MAP *def\_layer\_name svrf\_layer\_name*

### Parameters

- *def\_layer\_name*

A required parameter that specifies the DEF layer name.

- *svrf\_layer\_name*

A required parameter that specifies the original layer name or the derived layer name to which the DEF layer will be mapped.

### Description

This statement maps a DEF layer name to a layer name used in the SVRF rule file. The *svrf\_layer\_name* may be an original layer name or a derived layer name. An *svrf\_layer\_name* may only be mapped once. If the same *svrf\_layer\_name* is encountered in more than one PEX DEF Map statement, an error will be issued and extraction will stop.

A *def\_layer\_name* may only be mapped once. If the same *def\_layer\_name* is encountered in more than one PEX DEF Map statement, an error will be issued and extraction will stop.

This statement has overriding precedence over all other DEF layer mapping mechanisms, including the “\_FDI” suffix in the [Layer](#) statements.

All layer mappings recognized by extraction will be reported in the log file.

### Examples

To map a DEF layer to an SVRF layer, put the following statement in the rule file:

```
PEX DEF MAP m1 metal1  
PEX DEF MAP m2 metal2
```

## PEX Density Estimate

Parasitic extraction

**PEX DENSITY *layer* ESTIMATE *value***

### Parameters

- *layer*  
A required original layer.
- *value*  
A required floating-point local density estimate.

### Description

Specifies an estimated value of the local density of a layer in the layout. This is used in conjunction with thickness variation statements. PEX Density Estimate takes precedence over calculated local density ([PEX Density Window](#)) when both are present.

For a description of the Calibre xRC in-die variation process, including information on when and when not to use it, refer to the [\*Calibre xRC User's Manual\*](#).

### Examples

```
PEX DENSITY metall1 ESTIMATE 0.2000
```

# PEX Density Window

Parasitic extraction

**PEX DENSITY** *layer* { **WINDOW** *wx* [*wy*] [*weight\_value*]  
[TRUNCATE] | **BACKUP** | **WRAP** } ...

## Parameters

- ***layer***  
A required original layer.
- ***wx***  
A required value in microns that specifies the width of the window in the x direction. If *wy* is not specified, *wx* is used for both dimensions and the window is square.
- ***wy***  
An optional value in microns that specifies the height of the window in the y direction.
- ***weight\_value***  
An optional value that specifies a weight applied to a density window. Specify *weight\_value* between 0 and 1. The default value is 1.
- **TRUNCATE**  
An optional keyword that truncates partial windows to include only the extent of the design. This is the default behavior if you do not specify this keyword.
- **BACKUP**  
An optional keyword that specifies the last window be shifted back to align it right-hand or top edges with the design extent. This means it now overlaps the previous window.
- **WRAP**  
An optional keyword that specifies the design geometries be repeated beyond the design extent as if several instances of the design were tiled. The boundary window will include some of the geometries from the next instance of the design.

## Description

Specifies the window size, in microns, for density computation. The default window size is the cell size.

You can specify multiple window sizes. When specifying more than one window, use *wx*, *wy*, and *weight\_value*. The Calibre xRC tool calculates a layer density for each [*wx*, *wy*] pair. The weighted density is calculated as follows:

$$D = W_1 * D_{[wx_1, wy_1]} + W_2 * D_{[wx_2, wy_2]} + \dots + W_N * D_{[wx_N, wy_N]}$$

where:

- D is the total calculated density
- $W_N$  is the *weight\_value* factor for window N
- $D_{[wxN, wyN]}$  is the calculated density for the window specified by wx, wy

For a single window, wy and *weight\_value* are not required.

---

### **Caution**



Do not set the window size smaller than what is necessary. When the window size is set too small, to a few microns for example, this statement may use up all available memory.

---

Options TRUNCATE, BACKUP, or WRAP are used to control density calculations by allowing the user to specify boundary window handling similar to that used by the [Density](#) statement. Refer to the Density statement WINDOW option for diagrams of these modes.

For a description of the Calibre xRC in-die variation process, including information on when and when not to use it, refer to the [Calibre xRC User's Manual](#).

### Examples

The following statement specifies a density window that is 55 microns x 55 microns and uses TRUNCATE boundary mode:

```
PEX DENSITY metall1 WINDOW 55
```

The following statement specifies three separate density windows with different weights:

```
PEX DENSITY metall1 WINDOW 10 10 0.2 WINDOW 20 20 0.3 WINDOW 50 50 0.5
```

The following statement specifies two separate density windows with different specified weights. Use the default TRUNCATE boundary mode for the first window and BACKUP boundary mode for the second window:

```
PEX DENSITY metall1 WINDOW 10 10 0.4 WINDOW 20 20 0.6 BACKUP
```

The following statement specifies two separate density windows with different sizes and different specified weights. It also uses two different boundary modes for each window:

```
PEX DENSITY metall1 WINDOW 10 10 0.4 BACKUP WINDOW 20 20 0.6 WRAP
```

The following statement specifies a 10x10 density window with the default weight of 1.0 and BACKUP boundary mode:

```
PEX DENSITY metall1 WINDOW 10 10 BACKUP
```

## PEX Driver File

Parasitic extraction

### PEX DRIVER FILE *driver\_and\_receiver\_file\_location*

---

**Note**

 PEX Driver File has been renamed to [PEX Inductance Driver File](#). There are no changes to parameter definitions and functionality. PEX Driver File has been deprecated as of the 2009.1 release.

---

## PEX Elayer

Parasitic extraction

**PEX ELAYER** *layer\_tag* *layer1* *layer2* [*layerN* ...]

### Parameters

- ***layer\_tag***  
A required argument that identifies a particular PEX Elayer statement.
- ***layer1***  
A required argument that identifies the name of the layer from which the height and in-die variation settings are inherited by ***layer2*** and ***layerN***.
- ***layer2***  
A required argument that identifies the name of the layer to inherit the height and in-die variation settings.
- ***layerN***  
An optional argument that identifies the name of the layer(s) to inherit the height and in-die variation settings. You can specify ***layerN*** any number of times in one statement.

### Description

Defines the physical equivalence of layers. Layers inherit the height and in-die variation settings of the first layer. The settings do not overwrite any properties that the layer already had.

---

#### Note



If a rule file contains [PEX Alias](#), the compiler returns an error if it also includes a PEX Elayer statement.

---

### Examples

The following statement sets the geometries on the metal\_1, m1, and m1a layers to be physically equivalent during calculations on those layers.

```
PEX ELAYER m1_tag metal_1 m1 m1a
```

# PEX Exclude Distributed

Parasitic extraction

**Note**



As of the 2008.4 release, this statement has been deprecated. Use [PEX Extract Exclude](#) instead.

## PEX EXCLUDE DISTRIBUTED [LAYOUT | SOURCE] [TOPLEVEL | RECURSIVE]

*name* [*name* ...]

Used only in Calibre xRC.

### Parameters

- LAYOUT

An optional keyword that specifies the names are derived from the layout. This is the default behavior if you do not include a choice from this set in the statement.

- SOURCE

An optional keyword that specifies the names are derived from the source.

- TOPLEVEL

An optional keyword that specifies that the search is to take place in the top level. This is the default behavior if you do not include a choice from this set.

- RECURSIVE

An optional keyword that specifies that the search is to take place on all levels.

- ***name***

A required net to be excluded from RC extraction. You can specify ***name*** any number of times in one statement. The ***name*** can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Names that include wildcards must be enclosed in quotation marks.

### Description

Specifies to limit distributed RC extraction to the net names *not* listed. That is, specified nets are excluded from processing.

The excluded net names can include the question mark character (?) as a wildcard. The tool supports wildcards in the net name only, not in the path. For example, “X0/X1/foo?” is supported, but “?/X0/vdd” is not supported. In other words, the wildcard character does not match hierarchy. Names that include wildcards must be enclosed in quotation marks.

The rule file compiler will fail if you attempt to use the TOPLEVEL keyword with wildcards in the path. The rule file compiler will also fail if you use the RECURSIVE keyword with pathnames used to specify a specific net.

## **Examples**

```
PEX EXCLUDE DISTRIBUTED vcc vss
```

```
PEX EXCLUDE DISTRIBUTED LAYOUT "?vdd?"
```

# PEX Exclude Lumped

Parasitic extraction

**Note**



As of the 2008.4 release, this statement has been deprecated. Use [PEX Extract Exclude](#) instead.

**PEX EXCLUDE LUMPED** [LAYOUT | SOURCE] [TOPLEVEL | RECURSIVE] *name*  
[*name...*]

Used only in Calibre xRC.

## Parameters

- LAYOUT

An optional keyword that specifies the names are derived from the layout. This is the default behavior if you do not include a choice from this set in the statement.

- SOURCE

An optional keyword that specifies the names are derived from the source.

- TOPLEVEL

An optional keyword that specifies that the search is to take place in the top level. This is the default behavior if you do not include a choice from this set.

- RECURSIVE

An optional keyword that specifies that the search is to take place on all levels.

- ***name***

A required net to be excluded from lumped C extraction. You can specify ***name*** any number of times in one statement. The ***name*** can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Names that include wildcards must be enclosed in quotation marks.

## Description

Specifies to limit lumped C extraction to the net names *not* listed. That is, specified nets are excluded from processing.

Multiple statements form a single combined set.

The excluded net names can include the question mark character (?) as a wildcard. The tool supports wildcards in the net name only, not in the path. For example, “X0/X1/fo??” is supported, but “?/X0/vdd” is not supported. In other words, the wildcard character does not match hierarchy. Names that include wildcards must be enclosed in quotation marks.

The rule file compiler will fail if you attempt to use the TOPLEVEL keyword with wildcards in the path. The rule file compiler will also fail if you use the RECURSIVE keyword with pathnames used to specify a specific net.

## **Examples**

```
PEX EXCLUDE LUMPED VCC VSS  
PEX EXCLUDE LUMPED LAYOUT "?vdd?"
```

## PEX Extract Exclude

Parasitic extraction

### **PEX EXTRACT EXCLUDE {LAYOUTNAMES | SOURCENAMES}**

[TOPLEVEL | RECURSIVE] *netname* [*netname* ...]

Used only in Calibre xRC.

#### Parameters

- **LAYOUTNAMES**

A required keyword that specifies the names are derived from the layout. This is the default behavior if you do not include a choice from this set in the statement.

- **SOURCENAMES**

An required keyword that specifies the names are derived from the source.

- **TOPLEVEL**

An optional keyword that specifies that the search is to take place in the top level. This is the default behavior if you do not include a choice from this set.

- **RECURSIVE**

An optional keyword that specifies that the search is to take place on all levels.

- ***netname***

A required name that specifies a net to be excluded from extraction. You can specify *netname* any number of times in one statement. The *netname* can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Names that include wildcards must be enclosed in quotation marks.

#### Description

Specifies to limit extraction to the net names *not* listed.

The excluded net names can include the question mark character (?) as a wildcard. The Calibre xRC tool supports wildcards in the net name only, not in the path. For example, “X0/X1/vdd?” is supported, but “?/X0/vdd” is not supported. In other words, the wildcard character does not match hierarchy. Names that include wildcards must be enclosed in quotation marks.

The rule file compiler will fail if you attempt to use the TOPLEVEL keyword with wildcards in the path. The rule file compiler will also fail if you use the RECURSIVE keyword with pathnames used to specify a specific net.

An excluded net is not removed from the layout database during extraction. The excluded net is processed during extraction and parasitic coupling capacitances from the other nets to the excluded nets are included in the output netlist as lumped capacitors to ground. The parasitics for the excluded nets are not included in the output netlist.

When coupling capacitance is computed between two nets, the capacitance values from each net are taken into consideration. Excluding a net means that it’s capacitance will not be calculated

during extraction to determine the resulting values. The coupling capacitance to an excluded net is computed from only the included net's perspective. For information on how to capture coupling effects for all nets, see [PEX Netlist Select File](#).

### Examples

#### Example 1

To exclude nets from extraction with layout names vcc and vss, include the following SVRF statement in your rule file:

```
PEX EXTRACT EXCLUDE vcc vss
```

#### Example 2

To exclude all nets from extraction with source netlist names that contain vdd, include the following SVRF statement in your rule file:

```
PEX EXTRACT EXCLUDE SOURCENAMES "?vdd?"
```

# PEX Extract Floating Nets

Parasitic extraction

**PEX EXTRACT FLOATING NETS {GROUNDED [SCALE *factor*] | **ALL** | **REDUCED**} [IGNORE\_PORT\_TEXT]**

## Parameters

- **GROUNDED**

Keyword that specifies floating nets are treated as grounded when the capacitance for interacting nets is calculated. The capacitive effect on signal nets is shown as intrinsic capacitors on the signal net in the netlist. This is the default behavior.

- **SCALE *factor***

Optional keyword set used with GROUNDED to scale the capacitance values between floating and signal nets. The *factor* value must be greater than 0. The default value is 1.0.

- **ALL**

Keyword that specifies to extract and netlist all floating nets, including all coupling capacitors between floating polygons.

- **REDUCED**

Keyword that specifies that a floating net should be treated as floating when calculating its effects on signal nets. The floating net will not be netlisted. Only its effects will be included on the signal net.

- **IGNORE\_PORT\_TEXT**

Optional keyword that specifies to ignore port text placed on floating nets during extraction.

## Description

An optional statement that specifies how floating nets such as metal fill are to be handled during extraction. If the statement is not part of the rule file, floating nets are treated as grounded during extraction and not listed in the netlist.

Floating nets are defined as metal shapes that are not connected to devices, do not have port text, and are not defined as pins. A net is also considered floating if it has port text, but the IGNORE\_PORT\_TEXT parameter is specified.

Normally, the Calibre xRC tool does not netlist floating nets. By default, coupled capacitance between signal and floating nets is treated as intrinsic capacitance on the signal net. This is reported as part of the single intrinsic capacitance value for the net.

When REDUCED is specified for lumped or distributed RCC extraction, netlist size may increase, sometimes significantly, due to the introduction of new coupling capacitors. If the netlist is too large to simulate, reduce the number of parasitic capacitors with the [PEX Reduce CC](#) statement. If REDUCED is specified with selected net extraction, the floating nets around the selected nets must also be selected or they are assumed to be grounded.

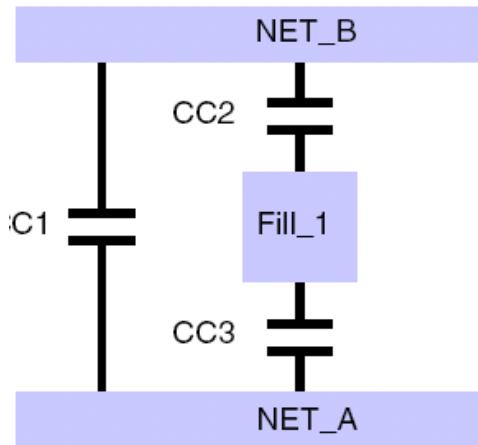
## Examples

### Example 1

This example shows how applying PEX Extract Floating Nets controls extracting metal fill. To simplify the pictures, extracted intrinsic capacitors are not shown. Effective capacitors to ground (\_Eff) represent the effects of the metal fill on signal nets.

[Figure 4-254](#) shows a simplified layout for metal fill between two nets. [Figure 4-255](#) shows the resulting networks for different parameters when using RCC extraction (-rcc).

**Figure 4-254. Simple Floating Net Example**



[Figure 4-255](#) (a) shows the result of specifying the GROUNDED keyword in the SVRF statement:

```
PEX Extract Floating Nets GROUNDED
```

CC1 is the coupling capacitance between NET\_A and NET\_B. CC2 and CC3 occur as coupling capacitances between NET\_B/Fill\_1 and NET\_A/Fill\_1, respectively. However, both are grounded. The Fill\_1 net is not netlisted.

[Figure 4-255](#) (b) shows the result of specifying the ALL keyword as follows:

```
PEX Extract Floating Nets ALL
```

In this case, CC2 and CC3 are included as coupling capacitances to Fill\_1 net.

[Figure 4-255](#) (c) shows the result of specifying the REDUCED keyword as follows:

```
PEX Extract Floating Nets REDUCED
```

C\_EffA and C\_EffB represent the effective intrinsic capacitances to NET\_A and NET\_B, respectively. CC\_Eff is the effective coupling capacitance of the floating metal. It represents the total effect of CC2 and CC3, appropriately scaled.

**Figure 4-255. Extraction with Resistance and Coupled Capacitance (-rcc)**

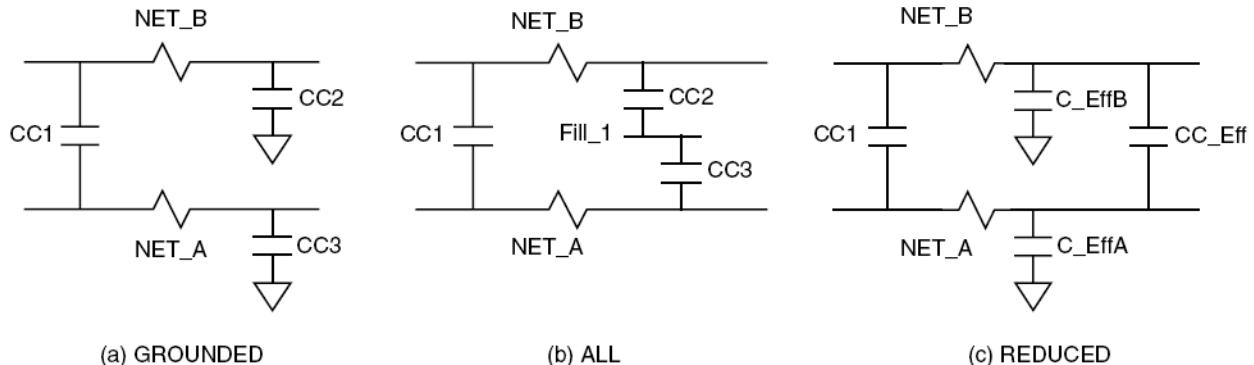


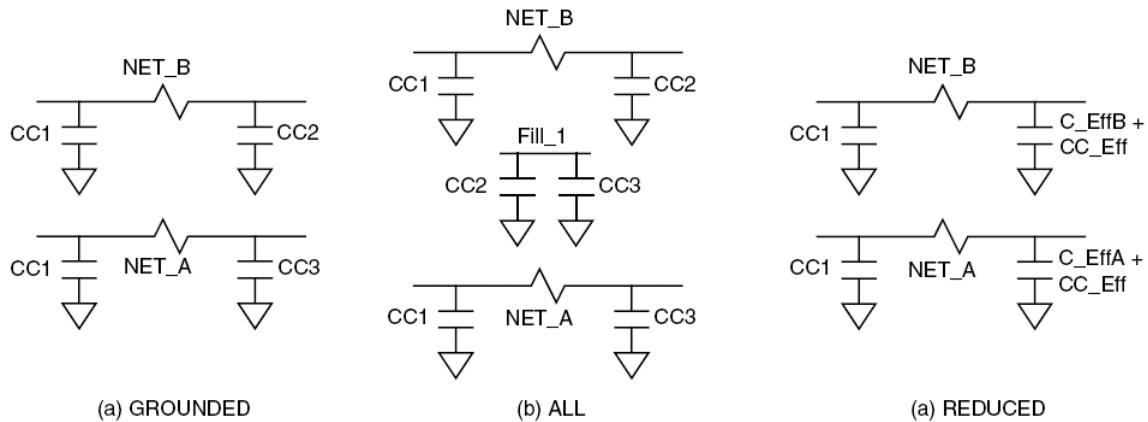
Figure 4-256 shows the resulting networks when using RC extraction (-rc).

Figure 4-256 (a) shows that the network includes only grounded capacitors when specifying the GROUNDED keyword.

Figure 4-256 (b) shows the result of specifying the ALL keyword. In this case, the coupling capacitors CC2 and CC3 are connected between Fill\_1 and ground.

Figure 4-256 (c) shows the result of specifying the REDUCED keyword. CC1 is added to both NET\_A and NET\_B as a capacitor to ground. CC\_Eff is the combination of CC2 and CC3 and is added to C\_EffA and C\_EffB, which are the effective capacitors for each net, respectively.

**Figure 4-256. Extraction with Resistance and Capacitance (-rc)**



## Example 2

The following statement can be included in the rule file to specify extracting all parasitic capacitances and ignoring any port text for floating metal:

PEX EXTRACT FLOATING NETS ALL IGNORE\_PORT\_TEXT

### Example 3

To use a Miller scaling factor of 2.0 for grounded coupled capacitors, include the following statement in the rule file:

PEX EXTRACT FLOATING NETS GROUNDED SCALE 2.0

## PEX Extract Include

Parasitic extraction

**PEX EXTRACT INCLUDE { LAYOUTNAMES | SOURCENAMES }**  
[TOPLEVEL | RECURSIVE]  
*netname* [*netname* ...]

Used only in Calibre xRC.

### Parameters

- **LAYOUTNAMES**

Optional keyword that specifies the names are derived from the layout. This is the default behavior if you do not include a choice from this set in the statement.

- **SOURCENAMES**

Optional keyword that specifies the names are derived from the source.

- **TOPLEVEL**

Specifies that the search is to take place in the top level. This is the default behavior if you do not include a choice from this set.

- **RECURSIVE**

Specifies that the search is to take place on all levels.

- ***netname***

A required net to be included for extraction. You can specify ***netname*** any number of times in one statement. The ***netname*** can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Names that include wildcards must be enclosed in quotation marks.

### Description

Specifies to limit extraction to the listed net names. That is, only specified nets are processed. Multiple statements form a single combined set.

The net names can include the question mark character (?) as a wildcard. The tool supports wildcards in the net name only, not in the path. For example, “X0/X1/vin?” is supported, but “?/X0/vin” is not supported. In other words, the wildcard character does not match hierarchy. Names that include wildcards must be enclosed in quotation marks.

The rule file compiler will fail if you attempt to use the TOPLEVEL keyword with wildcards in the path. The rule file compiler will also fail if you use the RECURSIVE keyword with pathnames used to specify a specific net.

## Examples

### Example 1

To include nets from extraction with layout names in1 and net1, include the following SVRF statement in your rule file:

```
PEX EXTRACT INCLUDE in1 net1
```

### Example 2

To include nets from extraction with source netlist names that contain in and out, include the following SVRF statement in your rule file:

```
PEX EXTRACT INCLUDE SOURCENAMES RECURSIVE "?in?" "?out?"
```

## PEX Extract Rgate

Parasitic extraction

**PEX EXTRACT RGATE** *gatelayername* [*gatelayername* ...]  
[SINGLE *factor*] [DOUBLE *factor* [DELTA]]

### Parameters

- ***gatelayername***  
Required parameter that specifies the transistor gate layer. You can specify more than one gate layer name.
- **SINGLE *factor***  
Optional parameter that specifies the resistance adjustment factor for a transistor gate connected on one side. The default value is 0.3333.
- **DOUBLE *factor***  
Optional parameter that specifies the resistance adjustment factor for a transistor gate connected on both sides. The default value is 0.1667.
- **DELTA**  
Optional keyword that can be used with the DOUBLE parameter. When DELTA is specified, a negative resistor with half the value of the gate resistance ( $1/2 \times R_g$ ) is added to the network representing a double-sided gate connection.

### Description

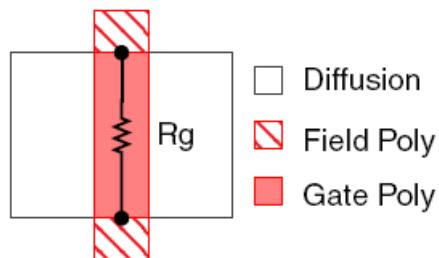
Specifies how to calculate the gate resistance of a transistor. This statement can be useful for differentiating between the resistance of gate and field polysilicon layers. Overlap of gate layer and interconnect polysilicon is not taken into account for the gate resistance calculation.

If you turn off gate resistance calculation in the simulation model for a MOSFET, PEX Extract Rgate can be used to extract this resistance. This statement is only effective for a rectangular-shaped gate area. Do not use the PEX Extract Rgate statement for non-standard shapes, such as a bent gate.

Only one PEX Extract Rgate statement is allowed in a rule file. If more than one statement exists, then extraction stops with an error message. Do not use PEX Extract Rgate with Resistance DEVICE\_SEED. This can result in a rule compilation error.

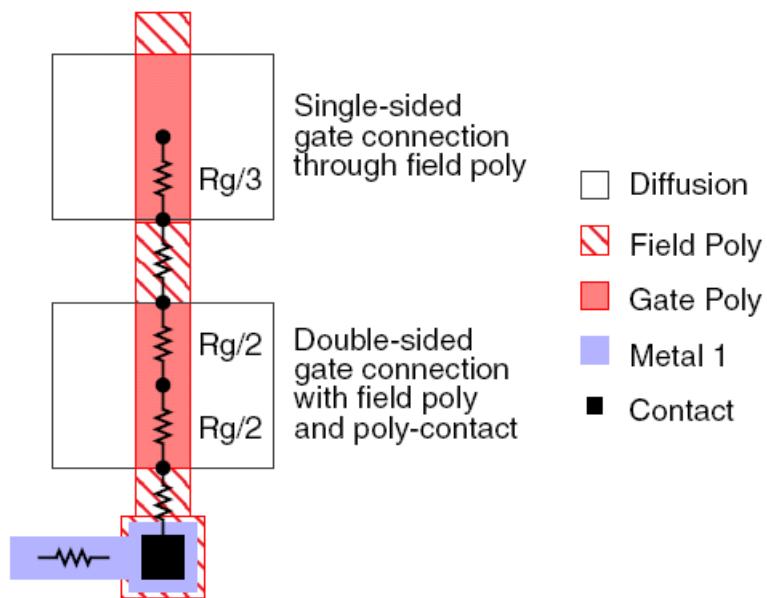
If you are using reduction statements, the gate resistors may be factored out of the netlist. For best results with PEX Extract Rgate, do not use reduction.

[Figure 4-257](#) shows an example of a valid gate area for this statement. Note that the field poly is connected to the gate poly and the only area considered by PEX Extract Rgate is the intersection of the gate poly and diffusion. This figure also shows the gate resistance calculated for an unconnected MOSFET, with a gate resistance value of  $R_g$ .

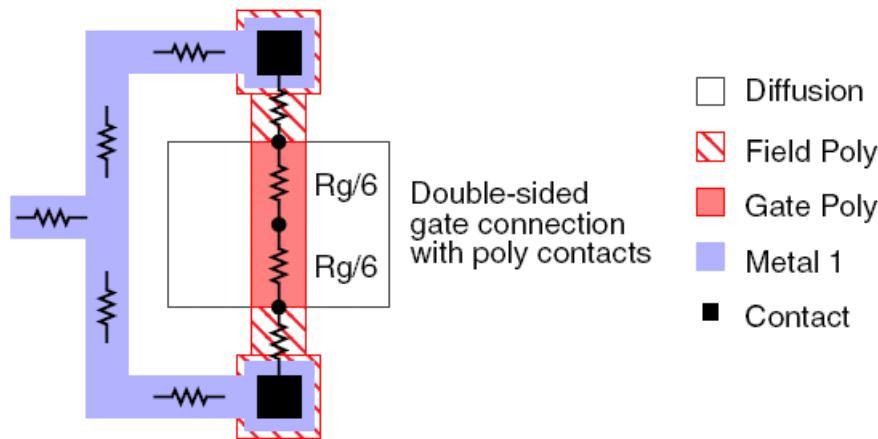
**Figure 4-257. MOSFET With Gate Resistor,  $R_g$ , and Valid Gate Shape**

For a single-sided connection, the value specified by the SINGLE parameter is applied to the gate resistance. This case is shown in [Figure 4-258](#).

A double-sided gate connection without an interconnect loop is also shown in [Figure 4-258](#). In this case, the gate is connected to a metal net on one side and another MOSFET on the other side. The gate resistance is split equally into two resistors.

**Figure 4-258. Single-Sided and Double-Sided Gate Connection**

[Figure 4-259](#) shows a double-sided gate connection with an interconnect loop, that is, both sides of the gate are connected to the same net. The gate resistance is split to two equal resistors and the value specified by the DOUBLE parameter is applied to each gate resistor.

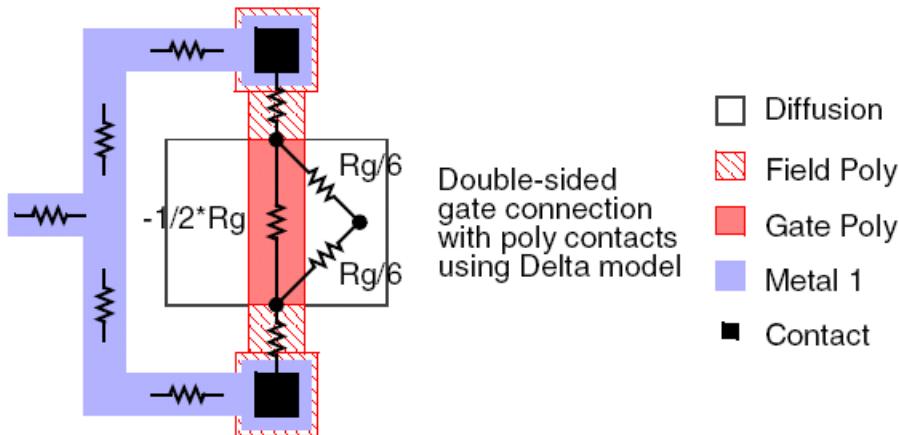
**Figure 4-259. Double-Sided Gate Connection With Interconnect Loop**

Double-sided  
gate connection  
with poly contacts

- Diffusion
- ▨ Field Poly
- Gate Poly
- Metal 1
- Contact

### Using the DELTA Parameter

The optional DELTA parameter may only be used with the DOUBLE parameter. When specified, the resulting network for a double-sided gate connection includes a negative resistor, as shown in [Figure 4-260](#).

**Figure 4-260. Double-Sided Gate Connection With Delta Model**

Double-sided  
gate connection  
with poly contacts  
using Delta model

- Diffusion
- ▨ Field Poly
- Gate Poly
- Metal 1
- Contact

#### Note



Many simulators do not handle negative resistance values. Do not use the DELTA option unless your simulator is able to handle negative resistance. This option may be used with circuit designs based on TSMC design kits.

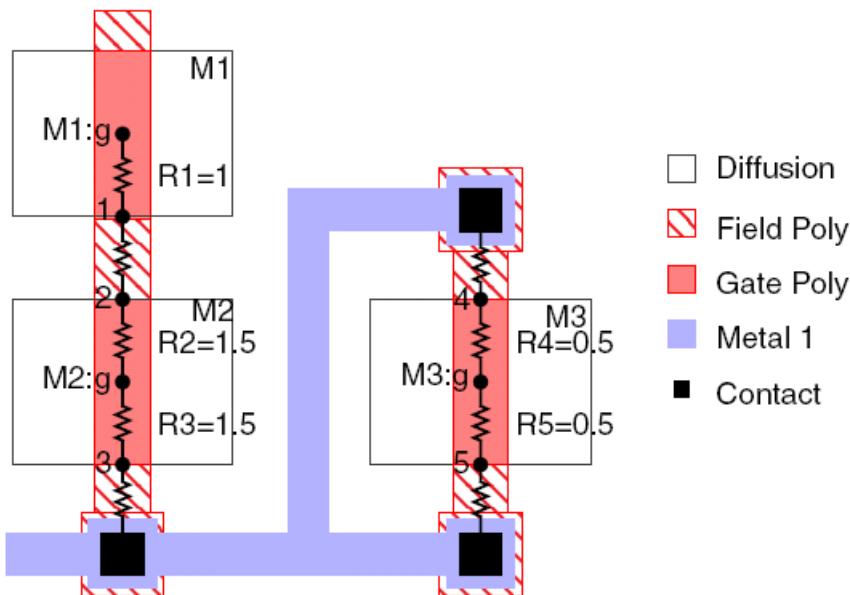
For more details on modeling reduced gate resistance, see William Liu, *MOSFET Models for Spice Simulation, Including BSIM3v3 and BSIM4*, 1st Edition, John Wiley and Sons, Inc., 2001.

## Examples

This example shows how PEX Extract Rgate is used to calculate gate resistances for transistors connected with different gate connections. All the gates are the same size and the gate resistance is 3 Ohms.

[Figure 4-261](#) shows three transistors, M1, M2, and M3, with gate connections as single-sided, double-sided without loop, and double-sided with loop respectively. To simplify the netlist, parasitic resistors for the metal 1 and field poly interconnect, poly contacts, and parasitic capacitances are not included in the output netlist.

**Figure 4-261. Gate Resistance Calculation Example**



The rule file contains the following SVRF statements:

```
// gate layer definition
gate = POLY and DIFF

// poly interconnect definition
poly_interconnect = POLY not gate

// to keep poly connectivity, connect gate area to poly interconnect
CONNECT gate poly_interconnect

// gate and poly_interconnect inherit the properties of the
// original layer and ignore the poly layer
PEX ALIAS POLY gate poly_interconnect IGNORE PRIMARY

// gate resistance extraction; uses default values
PEX EXTRACT RGATE gate
```

The netlist which contains the calculated gate resistors is as follows:

```
// M1: Single-sided gate connection
R1 M1:g 1 1
// M2: Double-sided gate connection to different nodes
R2 2 M2:g 1.5
R3 M2:g 3 1.5
// M3: Double-sided gate connection to same node
R4 4 M3:g 0.5
R5 M3:g 5 0.5
```

# PEX Extract Temperature

Parasitic extraction

**PEX EXTRACT TEMPERATURE *T* [NOMINAL *Tn*] [RANGE *T1 T2 T3*]**

## Parameters

- *T*  
Required floating-point number specifying the circuit temperature in degrees Celsius. The default value is 27 degrees.
- NOMINAL *Tn*

### Note



As of the 2010.3 release, NOMINAL has been deprecated and should not be used with calibrated rule files generated with the 2010.3 version or newer of xCalibrate. Calibrations and rule files generated with previous versions will continue to work.

Optional keyword and floating-point number specifying the nominal circuit temperature in degrees Celsius. The default value is 27 degrees.

- RANGE *T1 T2 T3*  
Optional keyword and three floating-point numbers specifying the three temperatures used in calculating TC1 and TC2 for [PEX Reduce ROnly](#) and [PEX Reduce Digital TICER](#) functionality. This functionality is enabled only when the [PEX Netlist](#) statement specifies the TCOEFF option. Default values for RANGE are -55, 27, and 155.

## Description

The PEX Extract Temperature statement allows you to modify the resistance extracted during the pdb step in the formatter by specifying the temperature dependence that can affect resistance. This statement is used in conjunction with the [Resistance Sheet](#) or [Resistance Rho](#) statement.

The temperature values are applied based on the following standard SPICE equation. (Note that TC1 and TC2 are defined for each layer with one of the statements [Parasitic Variation](#), [PEX Resistance Parameters](#), or [Resistance Sheet](#) in order of priority):

$$R = R * (1 + TC1 * (T - Tn) + TC2 * (T - Tn)^2)$$

The equation is applied during the formatting of the netlist, so you can modify temperature settings and reformat the netlist accordingly. Both the circuit temperature and the nominal temperature are written to the netlist, so that you can see what modifications were made to the parasitic components.

If PEX Extract Temperature is used but TC1 and TC2 are not specified for a layer, the tool issues an error message and exits.

## **PEX Extract Temperature**

---

### **Examples**

```
PEX EXTRACT TEMPERATURE 35
RESISTANCE SHEET M2 [.03 0 [0.0028 0.0035]]
RESISTANCE SHEET M1 [.02 0 [0.0027 0.0039]]
```

```
PEX EXTRACT TEMPERATURE 27 RANGE -55 27 155
```

# PEX Fieldsolver Boundary

Parasitic extraction

## PEX FIELDSOLVER BOUNDARY {X | Y} {REFLECTIVE | PERIODIC}

[BY\_EXTENT *min max* | BY\_ENCLOSURE *min max*]

Used only in Calibre xACT 3D.

### Parameters

- **X**

A required parameter that specifies the boundary of the cell is for horizontal orientations.

- **Y**

A required parameter that specifies the boundary of the cell is for vertical orientations.

- **REFLECTIVE**

A required parameter that specifies the placement of a mirror image of the geometries enclosed by the boundary on the other side of the boundary wall, and the same distance from it. A reflective boundary forces the normal component of the electric field at the boundary to be zero. May not be specified with PERIODIC.

- **PERIODIC**

A required parameter that specifies the replication of the geometries enclosed by the boundary on the other side of the boundary wall. A periodic boundary is different from a reflective boundary in that the geometries are copies rather than mirror images. May not be specified with REFLECTIVE.

- **BY\_EXTENT *min max***

An optional parameter set that specifies the absolute location of the boundary corresponding to the specified orientation, X or Y. The floating point values *min max* represent the minimum and maximum coordinate along the orientation in microns.

If neither BY\_EXTENT or BY\_ENCLOSURE are specified, the location of the boundary defaults to BY\_ENCLOSURE 0 0.

- **BY\_ENCLOSURE *min max***

An optional parameter set that specifies the distance of the boundary from the edge of the bounding box of metals in the cell along the specified orientation, X or Y. The floating point values *min* and *max* are in microns.

If neither BY\_EXTENT or BY\_ENCLOSURE are specified, the location of the boundary defaults to BY\_ENCLOSURE 0 0.

### Description

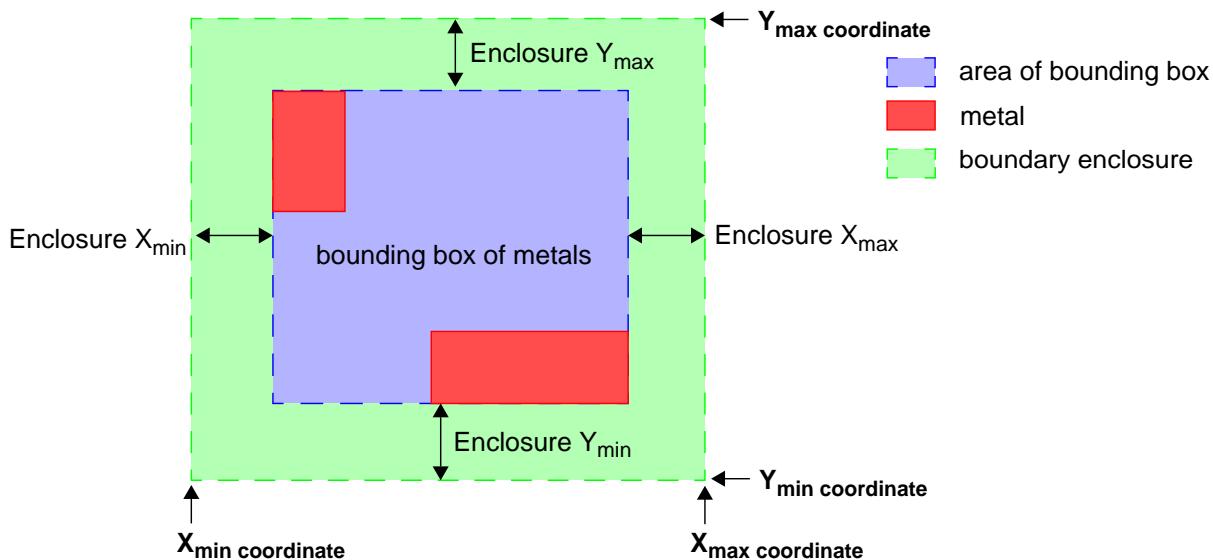
This optional statement specifies the boundary conditions used by the capacitance solver. Boundary conditions enable the extraction of an isolated cell by simulating the parasitic effects of neighboring conductors. This is useful for memory bit cell design to simulate various cell

placements and their effects on the memory cell geometries. Use this statement to improve extraction accuracy of memory cells or similar models. In the absence of this statement the surrounding space is treated as air.

Specify this statement separately for the horizontal (X) and vertical (Y) orientations. This statement can not be specified with [PEX Fieldsolver Cell\\_array](#) for the same orientation. For example, if your rule file contains a PEX Fieldsolver Cell\_array X statement, you can not specify a PEX Fieldsolver Boundary X statement.

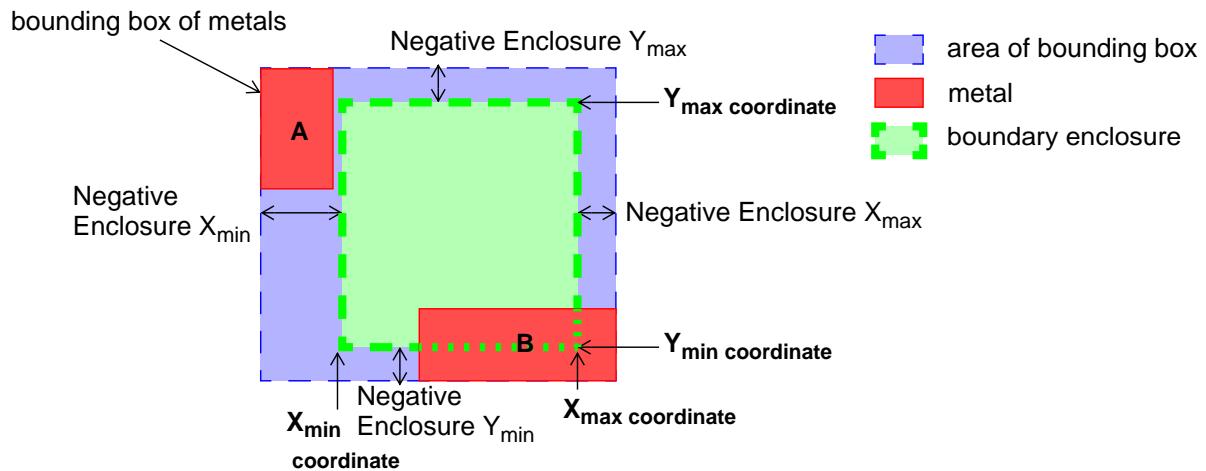
The boundary encloses the bounding box of the conductors (metal) in the cell. The size of the boundary is specified in terms of extent in each direction,  $X_{\min}$  and  $X_{\max}$  coordinates or  $Y_{\min}$  and  $Y_{\max}$  coordinates, or in terms of enclosure in the X or Y directions. [Figure 4-262](#) shows an example of what a bounding box area might look like for this statement. Note the colors do not represent different layer types other than metal, but are useful for viewing a pattern in the geometry. The default boundary location for BY\_ENCLOSURE 0 0 is the same as the bounding box of metals.

**Figure 4-262. Top View of Cell for Boundary**



When the boundary enclosure lies inside the bounding box of metals as shown in [Figure 4-263](#):

- Geometries entirely outside the boundary of the enclosure are not considered during capacitance calculations.
- Geometries partially outside the boundary enclosure are cut and only the portion found within the boundary is used.

**Figure 4-263. Top View of Cell for Internal Boundary**

## Examples

### Example 1

Include the following statements to specify the boundary type as reflective for X and periodic for Y:

```
PEX FIELDSOLVER BOUNDARY X REFLECTIVE
PEX FIELDSOLVER BOUNDARY Y PERIODIC
```

In this example, the boundary location is the same as the bounding box since neither BY\_EXTENT or BY\_ENCLOSURE were specified.

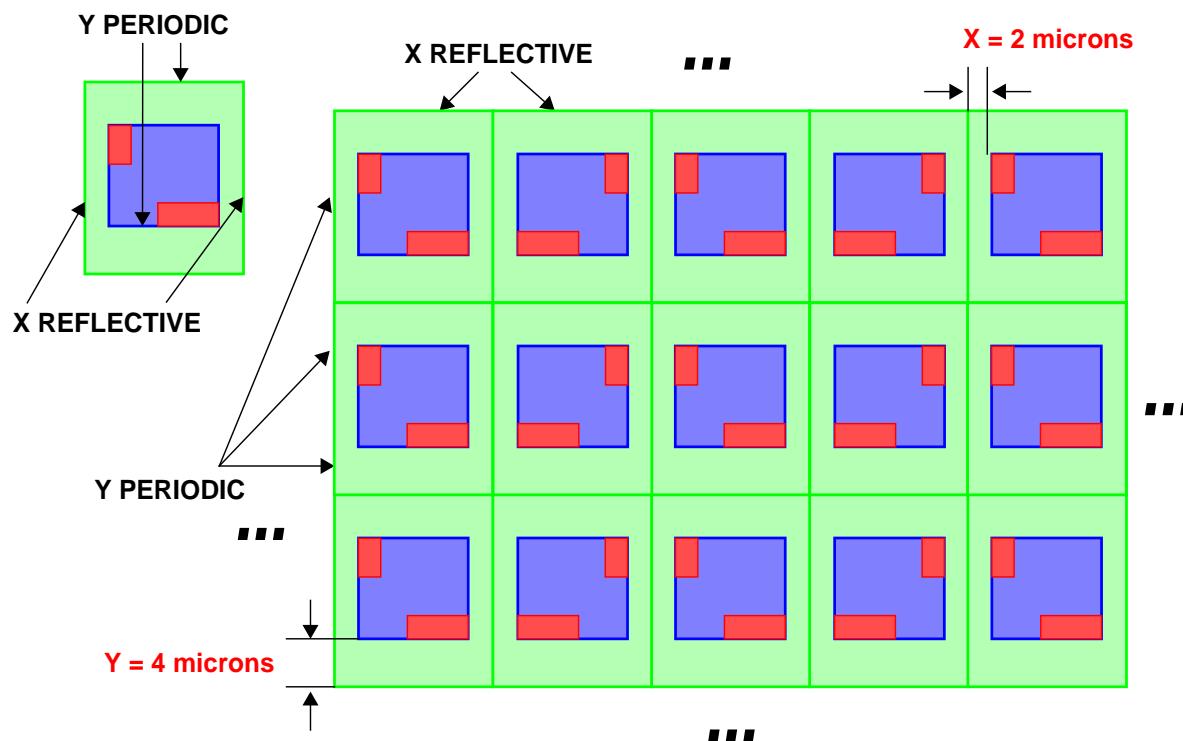
### Example 2

Include the following statements to specify the boundary type as reflective for X and periodic for Y:

```
PEX FIELDSOLVER BOUNDARY X REFLECTIVE BY_ENCLOSURE 2 2
PEX FIELDSOLVER BOUNDARY Y PERIODIC BY_ENCLOSURE 4 4
```

In this example the boundary location is defined by an enclosure of 2 microns in the x-direction and 4 microns in the y-direction outside the bounding box.

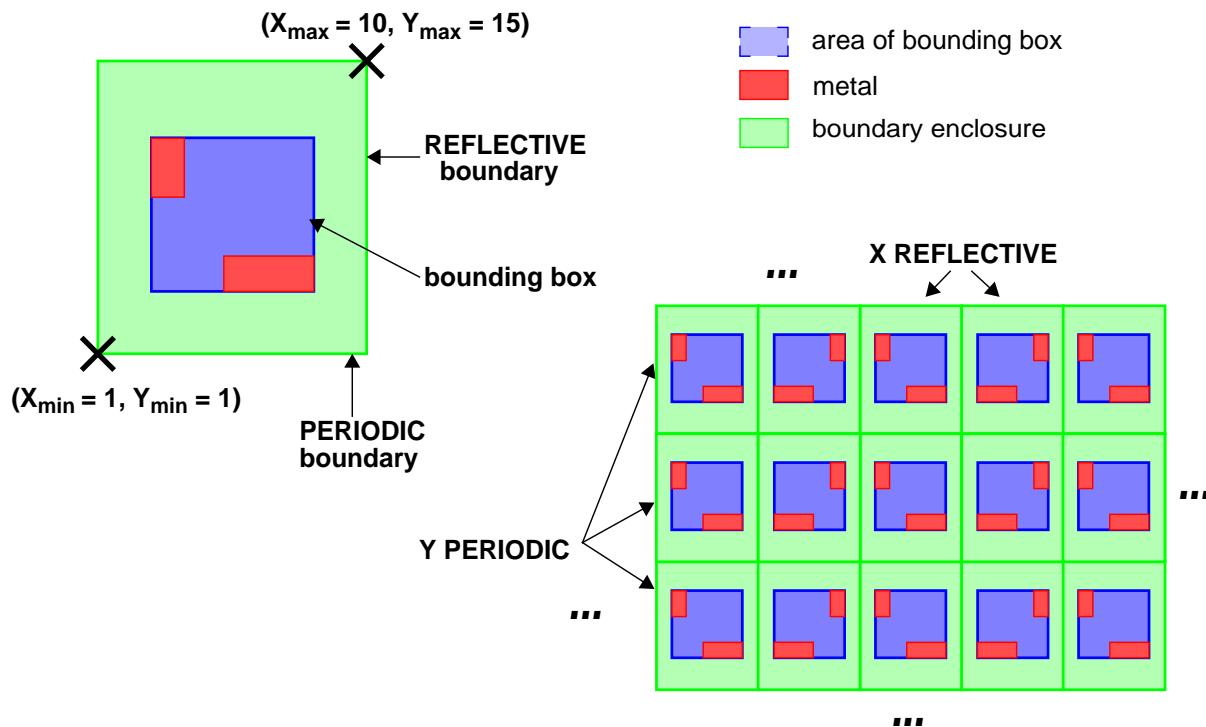
[Figure 4-264](#) shows an example of the boundary location using BY\_ENCLOSURE.

**Figure 4-264. Boundary Locations Using BY\_ENCLOSURE****Example 3**

Use the following statements to specify the boundary condition along the x-axis as a reflective cell and the y-axis as a periodic cell:

```
PEX FIELDSOLVER BOUNDARY X REFLECTIVE BY_EXTENT 1 10
PEX FIELDSOLVER BOUNDARY Y PERIODIC BY_EXTENT 1 15
```

In this example the boundary location in the x-direction is defined by absolute locations,  $X_{\min} = 1$  and  $X_{\max} = 10$ , along the x-axis. The boundary location in the y-direction is defined by absolute locations,  $Y_{\min} = 1$  and  $Y_{\max} = 15$ , along the y-axis. For a top view of the reflective and periodic boundary settings, see [Figure 4-265](#).

**Figure 4-265. BOUNDARY Locations Using BY\_EXTENT**

## PEX Fieldsolver Cell\_array

Parasitic extraction

### PEX FIELDSOLVER CELL\_ARRAY {X | Y} {REFLECTIVE | PERIODIC}

[BY\_EXTENT *min max* | BY\_ENCLOSURE *min max*] [OPEN\_BOUNDARY]

Used only in Calibre xACT 3D.

#### Parameters

- **X**

A required parameter that specifies the boundary of the cell is for horizontal orientations.

- **Y**

A required parameter that specifies the boundary of the cell is for vertical orientations.

- **REFLECTIVE**

A required parameter that specifies the placement of a mirror image of the enclosed geometries on the other side of the boundary, and grounds all conductors in the image. May not be specified with PERIODIC.

- **PERIODIC**

A required parameter that specifies the replication of the enclosed geometries on the other side of the boundary, and grounds all replicated conductors in the image. A periodic cell\_array is different from a reflective cell\_array in that the geometries are copies rather than mirror images. May not be specified with REFLECTIVE.

- **BY\_EXTENT *min max***

An optional parameter set that specifies the absolute location of the boundary corresponding to the specified orientation, X or Y. The floating point values *min max* represent the minimum and maximum coordinate along the orientation in microns.

If neither BY\_EXTENT or BY\_ENCLOSURE are specified, the location of the boundary defaults to BY\_ENCLOSURE 0 0.

- **BY\_ENCLOSURE *min max***

An optional parameter set that specifies the distance of the boundary from the edge of the bounding box of metals in the cell along the specified orientation, X or Y. The floating point values *min* and *max* are in microns.

If neither BY\_EXTENT or BY\_ENCLOSURE are specified, the location of the boundary defaults to BY\_ENCLOSURE 0 0.

- **OPEN\_BOUNDARY**

An optional parameter that specifies not to modify metal geometries that are partially or completely outside the boundaries of the cell.

## Description

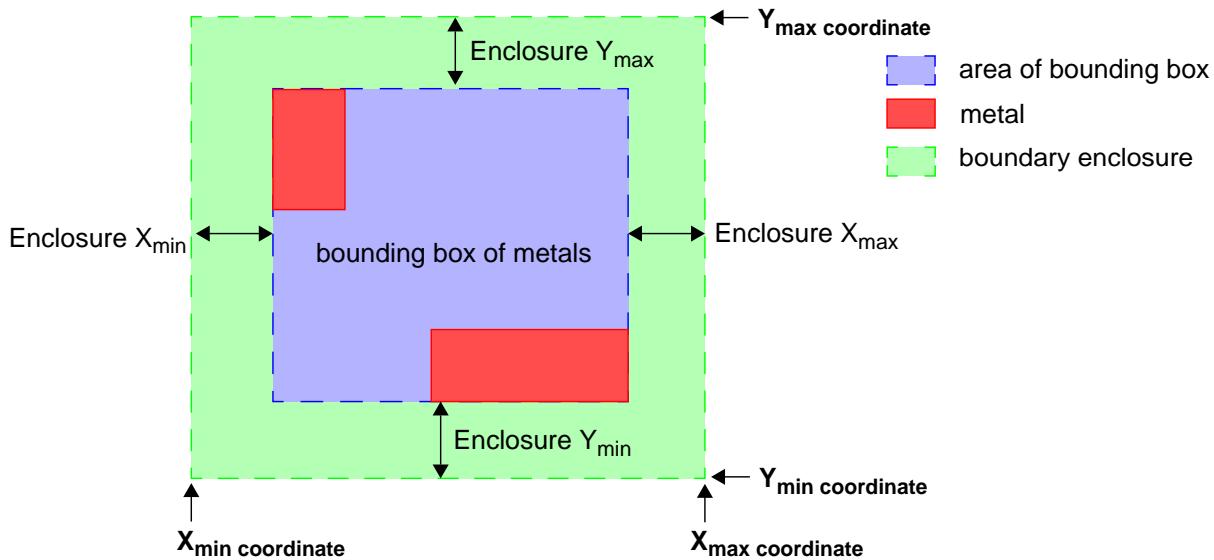
This optional statement specifies the boundary conditions used by the capacitance solver. All the conductors in the image that do not touch the boundary enclosure are grounded. Conductors that touch the boundary are treated as global nets.

Boundary conditions enable the extraction of an isolated cell by simulating the parasitic effects of neighboring conductors. Use this statement to improve extraction accuracy of memory cells or similar models. For example, this is useful for memory bit cell design when simulating cell placements and their effect on the memory cell geometries. Use this statement to improve extraction accuracy of memory cells or similar models. In the absence of this statement, the surrounding space is treated as air.

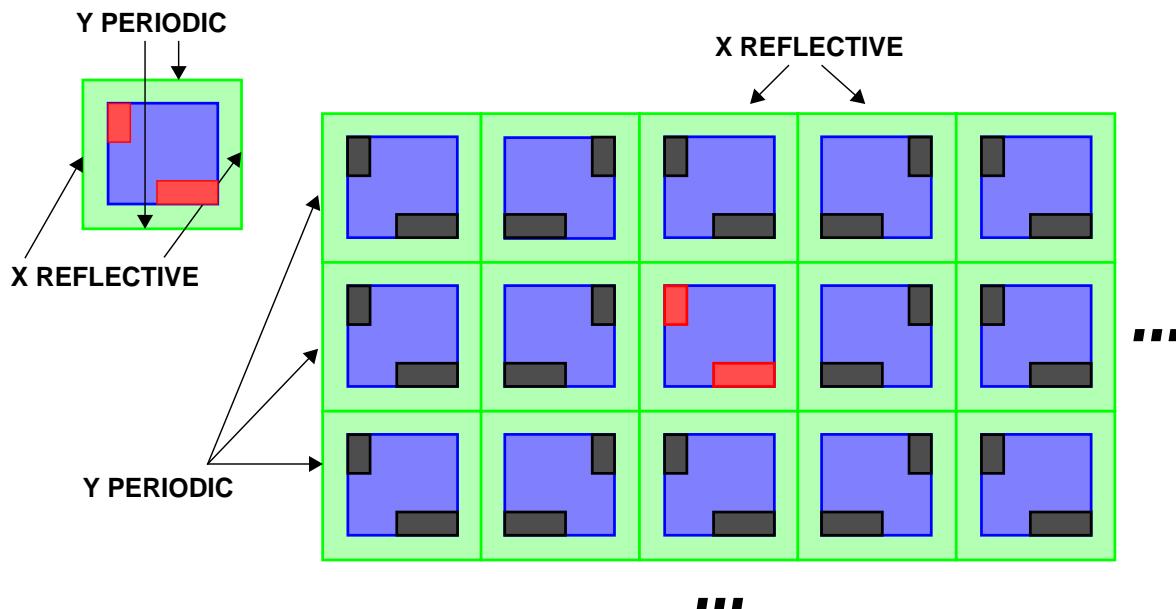
Specify this statement separately once for the horizontal (X) and vertical (Y) orientations. This statement can not be specified with [PEX Fieldsolver Boundary](#) for the same orientation. For example, if your rule file contains a PEX Fieldsolver Boundary X statement, you can not specify a PEX Fieldsolver Cell\_array X statement.

The boundary encloses the bounding box of the conductors in the cell. The size of the boundary is specified in terms of extent in each direction,  $X_{\min}$  and  $X_{\max}$  coordinates or  $Y_{\min}$  and  $Y_{\max}$  coordinates, or in terms of enclosure in the X or Y directions. [Figure 4-266](#) shows an example of what a bounding box area.

**Figure 4-266. Top View of Cell For Cell\_array**

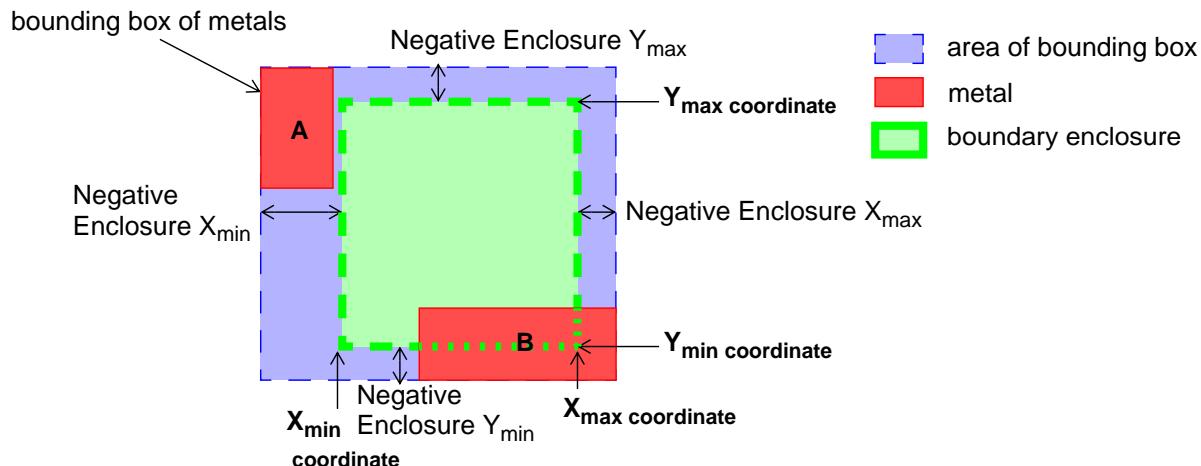


The colors do not represent different layer types other than metal, but are useful for viewing a pattern in the geometry. [Figure 4-267](#) shows an example of a cell array with reflective setting for the X orientation and periodic setting specified for the Y orientation. The tool determines the number of cells. In this example the boundary enclosure is larger (positive) than the bounding box of metals. Since none of the metals are touching the boundary enclosure, all of the metals in the replicated cells are grounded.

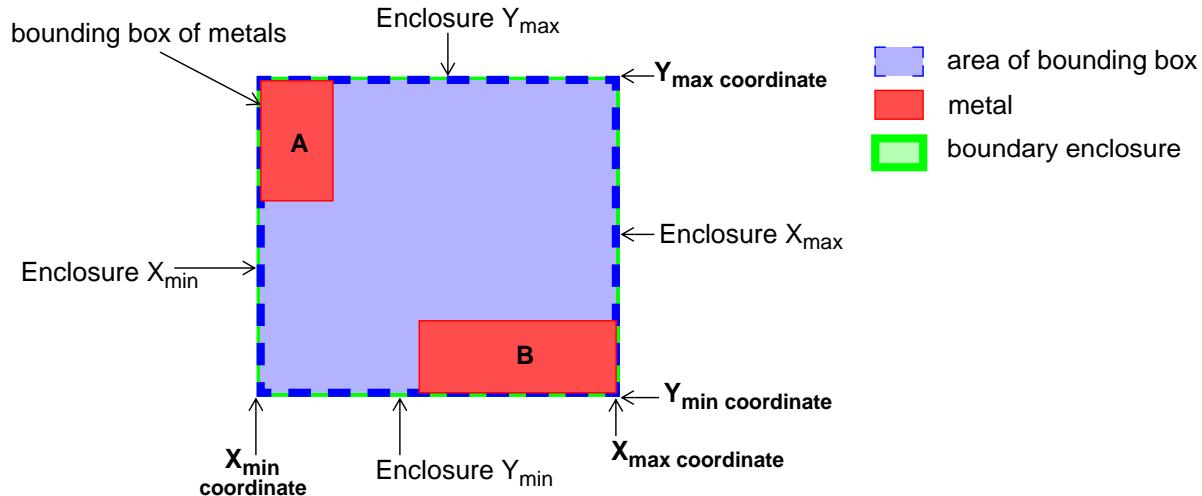
**Figure 4-267. Cell array Using REFLECTIVE(X) and PERIODIC(Y) Settings**

When the boundary enclosure lies inside the bounding box of metals as shown in [Figure 4-268](#):

- Geometries entirely outside the boundary of the enclosure are not considered during capacitance calculations. See the metal rectangle A.
- Geometries partially outside the boundary enclosure are cut and only the portion found within the boundary is used. See the metal rectangle B.

**Figure 4-268. Top View of Cell With Internal Boundary**

The default boundary location for BY\_ENCLOSURE 0 0 is the same as the bounding box of metals as shown in [Figure 4-269](#).

**Figure 4-269. Top View of Cell With Default Boundary**

## Examples

### Example 1

Include the following statements to specify the cell\_array type as reflective for X and periodic for Y, and ground all conductors in the image:

```
PEX FIELDSOLVER CELL_ARRAY X REFLECTIVE
PEX FIELDSOLVER CELL_ARRAY Y PERIODIC
```

In this example, all conductors in the x-direction mirror image are grounded and all the copied conductors in the y-direction are grounded. The boundary location is the same as the bounding box since neither BY\_EXTENT or BY\_ENCLOSURE were specified.

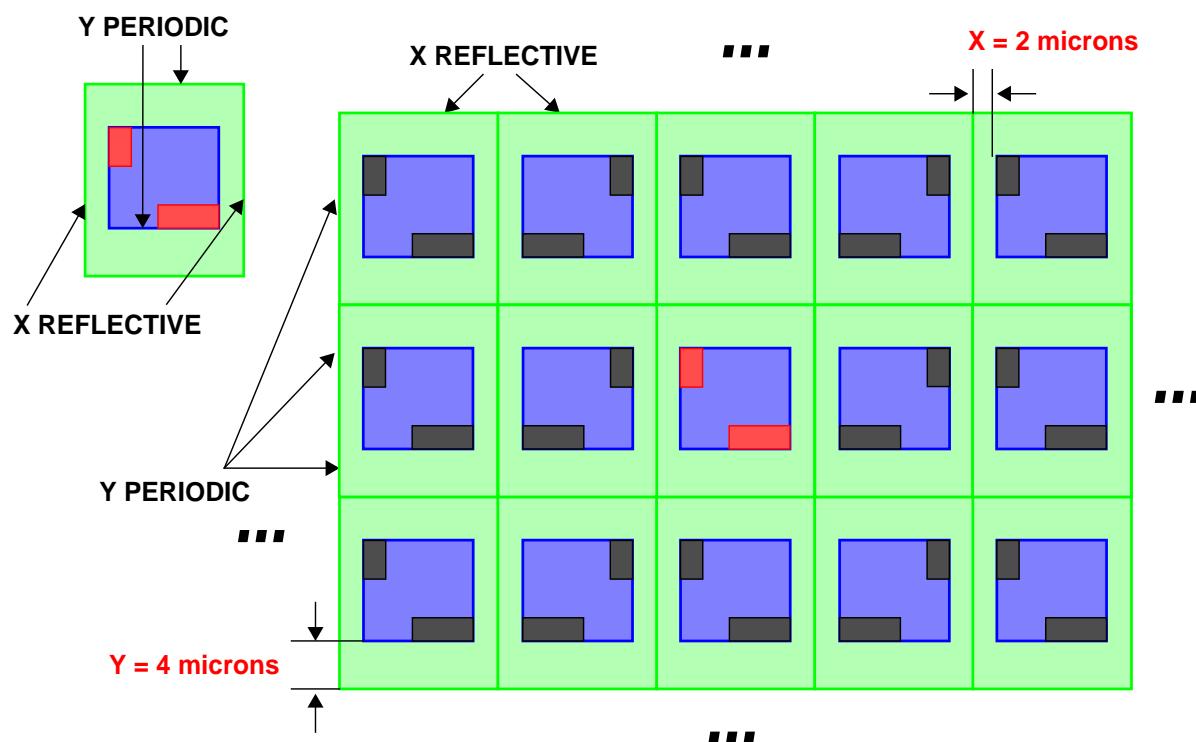
**Example 2**

Include the following statements to specify the boundary condition along the x-axis as a reflective cell and the y-axis as a periodic cell:

```
PEX FIELDSOLVER CELL_ARRAY X REFLECTIVE BY_ENCLOSURE 2 2
PEX FIELDSOLVER CELL_ARRAY Y PERIODIC BY_ENCLOSURE 4 4
```

In this example, the boundary location in the x-direction is defined by an enclosure of 2 microns, and the reflective images of the conductors are grounded. The boundary location in the y-direction is 4 microns, and the periodic images of the conductors are grounded. For a top view of the reflective and periodic boundary settings, see [Figure 4-270](#).

**Figure 4-270. Cell\_array Boundaries Using BY\_ENCLOSURE**



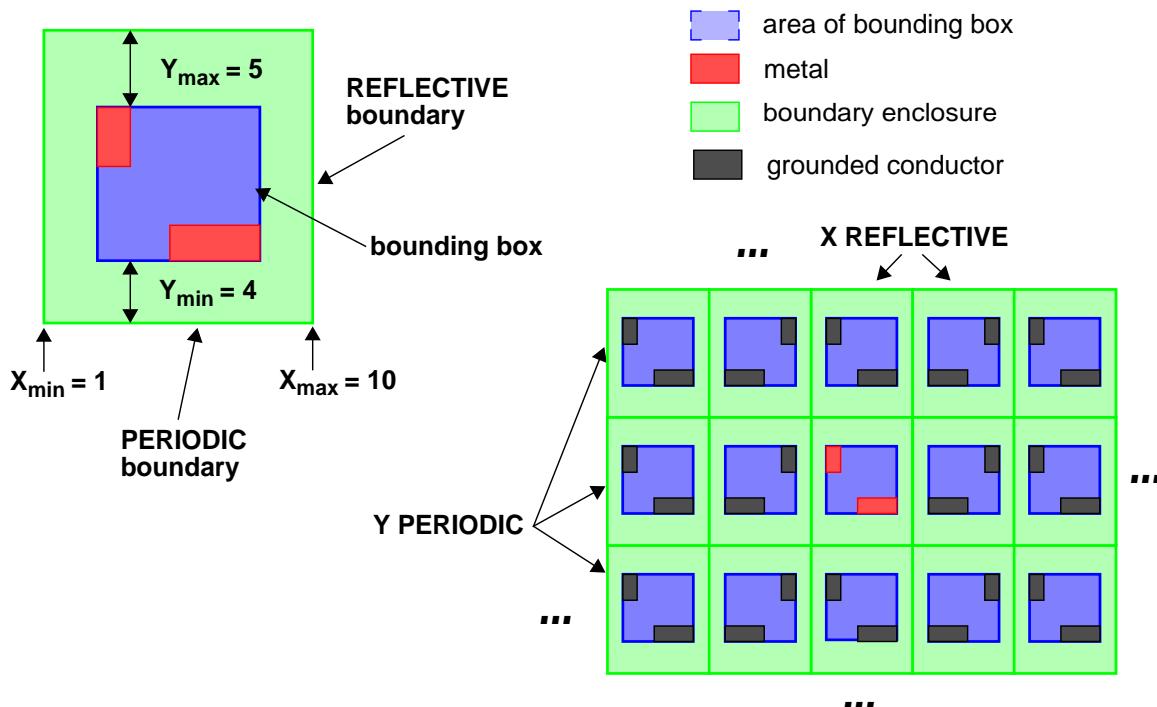
**Example 3**

Use the following statements to specify the boundary condition along the x-axis as a reflective cell and the y-axis as a periodic cell:

```
PEX FIELDSOLVER CELL_ARRAY X REFLECTIVE BY_EXTENT 1 10
PEX FIELDSOLVER CELL_ARRAY Y PERIODIC BY_ENCLOSURE 4 5
```

In this example the boundary location in the x-direction is defined by absolute locations,  $X_{\min} = 1$  and  $X_{\max} = 10$ , along the x-axis. The boundary location in the y-direction is 4 microns from the lower edge of the bounding box and 5 microns from the upper edge. All reflective and periodic images of the conductors are grounded. For a top view of the effect of these reflective and periodic boundary settings, see [Figure 4-271](#).

**Figure 4-271. Cell\_array Boundaries Using BY\_EXTENT and BY\_ENCLOSURE**



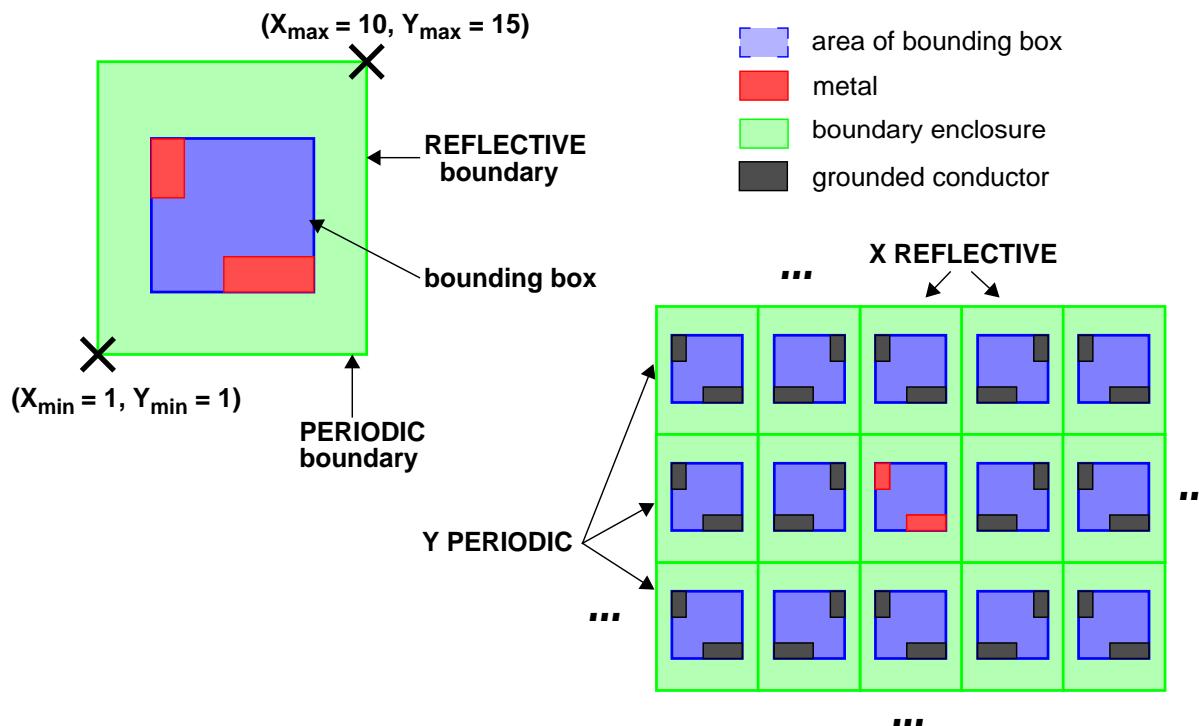
**Example 4**

Use the following statements to specify the boundary condition along the x-axis as a reflective cell and the y-axis as a periodic cell:

```
PEX FIELDSOLVER CELL_ARRAY X REFLECTIVE BY_EXTENT 1 10
PEX FIELDSOLVER CELL_ARRAY Y PERIODIC BY_EXTENT 1 15
```

In this example the boundary location in the x-direction is defined by absolute locations,  $X_{\min} = 1$  and  $X_{\max} = 10$ , along the x-axis. The boundary location in the y-direction is defined by absolute locations,  $Y_{\min} = 1$  and  $Y_{\max} = 15$ , along the y-axis. All reflective and periodic images of the conductors are grounded. For a top view of the effect of these reflective and periodic boundary settings, see [Figure 4-272](#).

**Figure 4-272. Cell\_array Boundaries Using BY\_EXTENT**



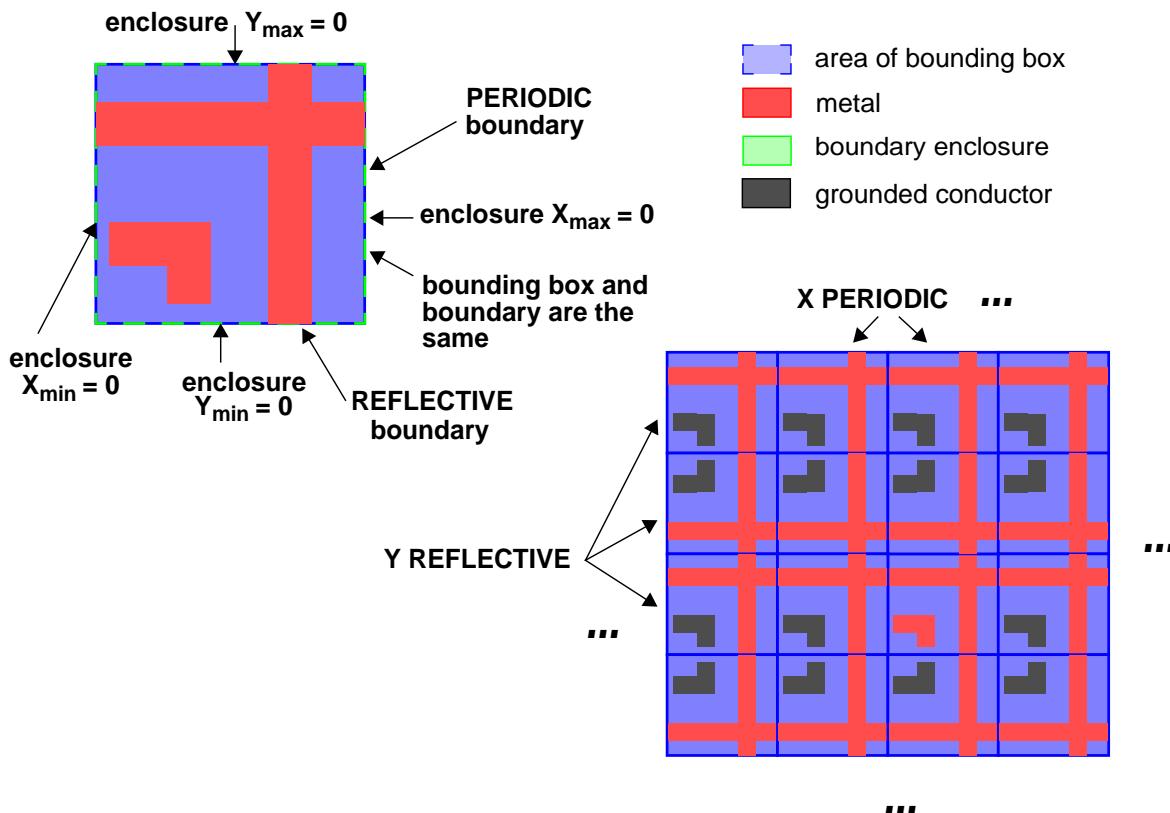
**Example 5**

Use the following statements to specify the boundary condition along the x-axis as a periodic cell and the y-axis as a reflective cell:

```
PEX FIELDSOLVER CELL_ARRAY X PERIODIC BY_ENCLOSURE 0 0
PEX FIELDSOLVER CELL_ARRAY Y REFLECTIVE BY_ENCLOSURE 0 0
```

In this example the boundary location in the x-direction is 0 microns from the left edge of the bounding box and 0 microns from the right edge. The boundary location in the y-direction is 0 microns from the lower edge of the bounding box and 0 microns from the upper edge. All reflective and periodic images of the conductors that do not touch the enclosure are grounded. For a top view of the effect of these reflective and periodic boundary settings, see [Figure 4-273](#).

**Figure 4-273. Cell\_array Boundaries With Floating Metal**



**Example 6**

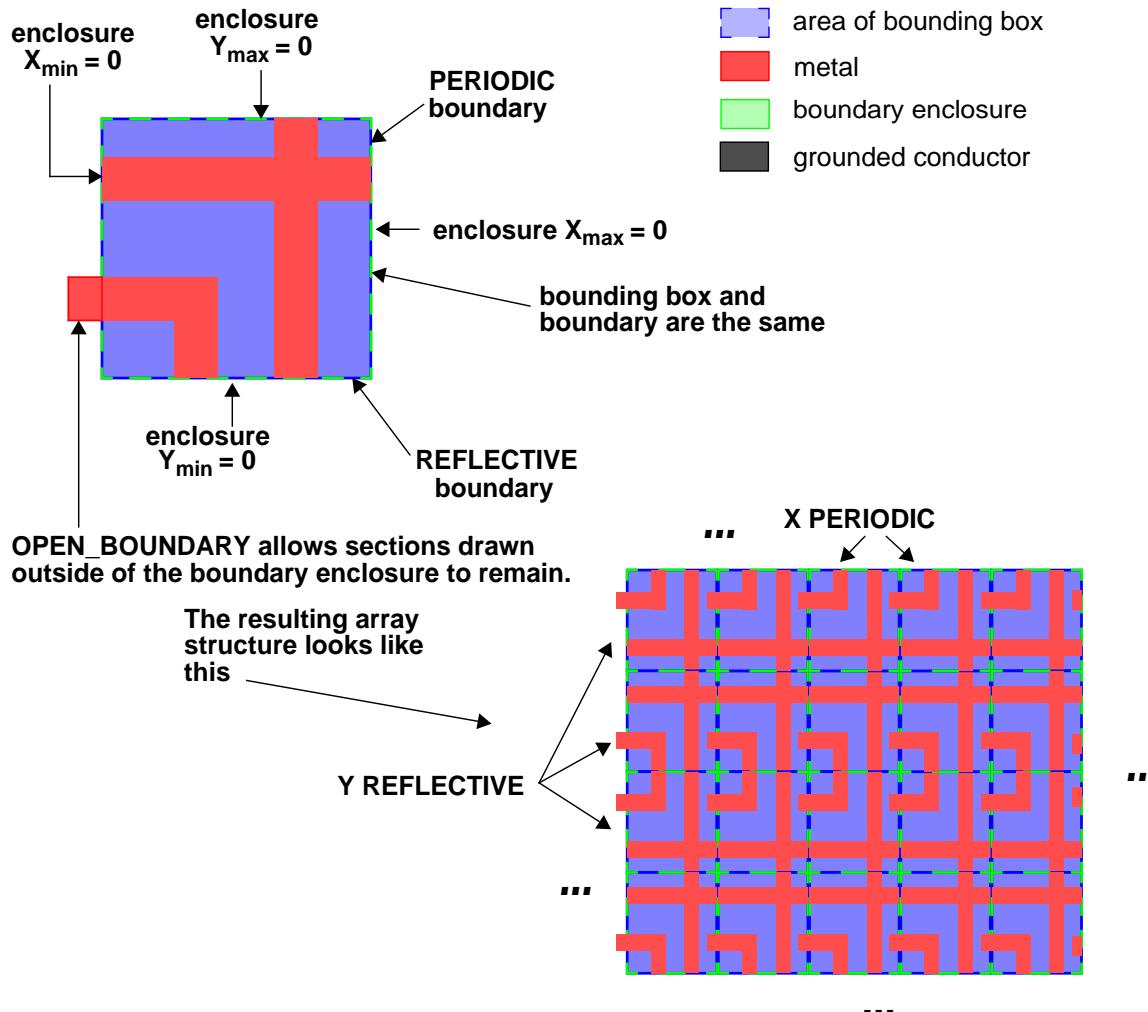
Use the following statements to specify the boundary condition along the x-axis as a periodic cell and the y-axis as a reflective cell:

```
PEX FIELDSOLVER CELL_ARRAY X PERIODIC BY_ENCLOSURE 0 0 OPEN_BOUNDARY
PEX FIELDSOLVER CELL_ARRAY Y REFLECTIVE BY_ENCLOSURE 0 0
```

In this example the boundary location in the x-direction is 0 microns from the left edge of the bounding box and 0 microns from the right edge. The boundary location in the y-direction is 0 microns from the lower edge of the bounding box and 0 microns from the upper edge.

By applying the OPEN\_BOUNDARY option in the x-direction, metal segments found outside the boundary enclosure of the cell are not cut. The cell shown in [Figure 4-274](#) contains a segment drawn outside the boundary. All reflective and periodic images of the conductors that do not touch the enclosure are grounded. For this example, all conductors are touching the boundary, so none of the images are grounded.

**Figure 4-274. Cell\_array Boundaries Using OPEN\_BOUNDARY**



# PEX Fieldsolver Endcap Spacing

Parasitic extraction

## PEX FIELDSOLVER ENDCAP SPACING *value*

Used only in Calibre xACT 3D.

### Parameters

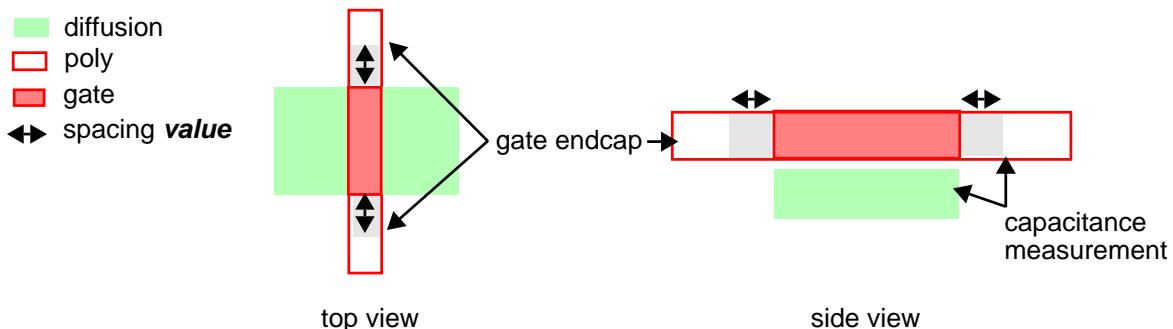
- *value*

A required floating point number that specifies a spacing width in user units. The default value is 0 if this statement is not specified.

### Description

An optional statement that allows you to specify a *virtual* spacing between the source/drain boundary and an endcap for fieldsolver device extraction. [Figure 4-275](#) shows the physical interpretation of the *value*.

**Figure 4-275. Endcap Spacing Distance**



A non-zero spacing *value* defines a virtual gap between the diffusion and the field poly. This is for the convenience of the field solver because the capacitance between the field poly and the diffusion can be more accurately modeled.

### Examples

Including the following statement should be used for 28 nm technologies and causes the Calibre xACT tool to use a different endcap spacing value in the fieldsolver capacitance extraction calculations:

```
// for 28 nm process
PEX FIELDSOLVER ENDCAP SPACING 0.018
```

## PEX Fieldsolver Mode

Parasitic extraction

**PEX FIELDSOLVER MODE {200 | 600} [{NETS | EXCLUDE} [LAYOUTNAMES | SOURCENAMES] [TOPLEVEL | RECURSIVE] *netname* [*netname* ...]]**

Used only in Calibre xACT 3D.

### Parameters

- **200**

Numerical value that sets the accuracy mode for the Calibre xACT tool. This is the default value.

- **600**

Numerical value that instructs the Calibre xACT tool to run at higher accuracy.

- **NETS**

Optional keyword that specifies the accuracy level used for extraction on one or more specified nets.

- **EXCLUDE**

Optional keyword that specifies the nets to be excluded from the Calibre xACT portion of a hybrid fieldsolver xACT 3D rule-based extraction run. This keyword may only be used with the -hybrid invocation switch.

- **LAYOUTNAMES**

Optional keyword that specifies the names are derived from the layout. This is the default behavior if you do not include a choice from this set in the statement.

- **SOURCENAMES**

Optional keyword that specifies the names are derived from the source.

- **TOPLEVEL**

Specifies that the search is to take place in the top level. This is the default behavior if you do not include a choice from this set.

- **RECURSIVE**

Specifies that the search is to take place on all levels.

- ***netname***

Specifies the net to be included for extraction. You can specify *netname* any number of times in one statement. The *netname* can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Names that include wildcards must be enclosed in quotation marks. May only be specified together with the NETS keyword.

## Description

An optional statement that allows you to set the capacitance extraction accuracy mode used by the Calibre xACT tool. Setting the accuracy mode to 600 is more accurate, but will significantly increase runtime. Specifying any value other than 200 or 600 will generate an error message and extraction will stop.

## Using NETS

The NETS option defines the extraction accuracy of specified nets when using [PEX Extract Include](#). PEX Fieldsolver Mode specified with the NETS option can appear more than once in your rule file. Without the NETS option, only one PEX Fieldsolver Mode statement can be specified.

Specifying a net more than once with different accuracy values results in a warning. The Calibre xACT tool applies the most accurate mode to the specified net. It is not an error to specify the accuracy mode for a net not being extracted.

If wildcards are used in the netname, then duplicate definitions may also occur. In this case the most accurate mode is applied to the nets.

The EXCLUDE option specifies which nets to exclude from extraction using the Calibre xACT tool. You must use the -hybrid invocation switch with this option. You can specify only one PEX Fieldsolver Mode statement with this option. Such a statement will be ignored for a regular Calibre xACT run.

## Examples

### Example 1

Include this statement to run the Calibre xACT tool with higher accuracy:

```
PEX FIELDSOLVER MODE 600
```

### Example 2

Include the following statements to control extraction accuracy for net\_a and net\_b:

```
PEX FIELDSOLVER MODE 600 NETS net_a
PEX FIELDSOLVER MODE 200 NETS net_b
```

In this example, net\_a is extracted at a higher level of accuracy than net\_b:

### Example 3

You can exclude large nets from fieldsolver extraction. For example, to apply rule-based extraction to supply nets only, include the following statement:

```
PEX FIELDSOLVER MODE 200 EXCLUDE VCC VSS
```

Invoke extraction as follows:

```
calibre -xact -3d -hybrid myrules
```

This applies fieldsolver extraction mode 200 to all nets in the design except VCC and VSS.

### Example 4

You can control which nets are extracted by Calibre xRC and Calibre xACT. For example, specify the following statements:

```
PEX EXTRACT EXCLUDE VCC VSS  
PEX FIELDSOLVER MODE 600 NETS IN?
```

Invoke extraction as follows:

```
calibre -xact -3d -hybrid myrules
```

Calibre xRC extracts all nets except VCC and VSS. Calibre xACT 3D extracts all IN? nets with fieldsolver extraction mode 600.

### Example 5

You can extract selected nets with Calibre xRC and then a subset of nets with Calibre xACT. For example, specify the following statements:

```
PEX EXTRACT INCLUDE IN?  
PEX FIELDSOLVER MODE 200 NETS IN1
```

Invoke extraction as follows:

```
calibre -xact -3d -hybrid -select myrules
```

Calibre xRC extracts all IN? nets, then the IN1 net is extracted with fieldsolver accuracy by Calibre xACT 3D.

# PEX Fill Handling

Parasitic extraction

**Note**



As of the 2008.3 release, this statement has been deprecated. Use [PEX Extract Floating Nets](#) instead.

## PEX FILL HANDLING {**GROUNDED** [*scale*] | **FLOATING** [*scale*] | **EXTRACT**}

### Parameters

- **GROUNDED** [*scale*]

Keyword and optional parameter that specifies the floating nets are treated as grounded when the capacitance for interacting nets is calculated. The coupled capacitance is then grounded for netlisting. This is the default.

The *scale* parameter is an optional positive floating point number used with GROUNDED or FLOATING to scale capacitance values between floating and signal nets. The default value is 1.0.

- **FLOATING**

Keyword that specifies that the floating net should be treated as floating when calculating its effects on signal nets. This setting cannot be used with the -cselect invocation switch.

- **EXTRACT**

Keyword that specifies to extract and netlist floating nets.

### Description

An optional statement that specifies how floating nets such as metal fill are to be handled during extraction. If the statement is not part of the rule file, floating nets are treated as grounded during extraction and not listed in the netlist.

A floating net is defined as a net that is not connected to a device and does not have port text. With the default behavior, which is the same as specifying GROUNDED, coupled capacitance between signal and floating nets is treated as intrinsic capacitance on the signal net. This is reported as part of the single intrinsic capacitance value for the net.

When FLOATING is specified for lumped or distributed RCC extraction, netlist size may increase, sometimes significantly, due to the introduction of new coupling capacitors. If the netlist is too large to simulate, reduce the number of parasitic capacitors with the [PEX Reduce CC](#) statement. If FLOATING is specified with selected net extraction, the floating nets around the selected nets must also be selected or they are assumed to be grounded.

See the “[Ignoring or Extracting Floating Nets](#)” section of the [xRC User’s Manual](#) for more information.

### Examples

```
PEX FILL HANDLING GROUNDED
```

## PEX Fill Model

Parasitic extraction

**PEX FILL MODEL {NO | {*layer\_name* *fill\_ratio* *fill\_spacing* *fill\_width* *fill\_type*}}**

### Parameters

- **NO**

Disables all virtual fill functionality for all layers. Overrides settings from other PEX FILL MODEL statements as well as the foundry fill model.

- ***layer\_name***

A required layer name identifying the conductor layer.

- ***fill\_ratio***

A required value that represents the fill density of the layer. It is a floating point value in a range  $0 \leq x < 1$ . The ***fill\_ratio*** also defines the minimal density of the layer. If calculated density at any location on the layer is less than ***fill\_ratio***, the value of ***fill\_ratio*** will be used. If calculated density is greater than the ***fill\_ratio***, ***fill\_ratio*** is ignored. This parameter will not be taken into account if you specify PEX DENSITY ESTIMATE for a layer.

- ***fill\_spacing***

A non-negative floating point value representing distance in user units. The value represents the minimum distance between routing and fill.

- ***fill\_width***

A required non-negative floating point value, currently not used by xRC. Use a value of 0 as a placeholder since a value must be specified.

- ***fill\_type***

A required non-negative integer value, currently not used by xRC. Use a value of 0 as a placeholder since a value must be specified.

### Description

An optional statement that allows you to override foundry fill model settings for a specified layer.

### Examples

```
PEX FILL MODEL metall 0.3 0.06 0 0
```

# PEX Fracture Frequency

Parasitic extraction

## **PEX FRACTURE [layer... ] FREQUENCY *hertz***

Used in Calibre xRC.

### Parameters

- *layer*  
An optional layer name or list of layer names on which the specified fracture frequency is applied. If a layer name is not specified then the frequency is applied to all layers.
- ***hertz***  
A required positive value that specifies the frequency in hertz. The format is a floating-point number expressed in scientific notation.

### Description

Instructs the Calibre xRC tool to fracture long straight wires and insert nodes more frequently using electrical considerations rather than geometric considerations. Using this statement will increase the number of resistors in the output netlist.

This statement may be specified more than once in the rule file, allowing frequencies to be specified for each layer. Refer to Example 3 in the Examples section of this statement. Note that only one layer-specific statement is allowed per layer and only one global setting statement is allowed in the rule file. Specifying multiple statements for the same layer will generate an error.

### Examples

#### Example 1

To specify the fracture frequency for all layers add the following line to your rule file:

```
PEX FRACTURE FREQUENCY 1.2e9
```

#### Example 2

To specify the fracture frequency for layers metal1 and metal2, add the following line to your rule file:

```
PEX FRACTURE metal1 metal2 FREQUENCY 1.3e9
```

#### Example 3

If both of the following statements appear in the rule file:

```
PEX FRACTURE FREQUENCY 1.2e9
// global frequency setting
PEX FRACTURE metal1 metal2 FREQUENCY 1.3e9
// layer specific frequency setting
```

the fracture frequency of 1.3 GHz will be used for metal1 and metal2. The fracture frequency of 1.2 GHz will be used for all other layers.

## PEX Generate Driver File Tag

Parasitic extraction

### PEX GENERATE DRIVER FILE TAG [*tags*]

Used in Calibre xRC.

#### Parameters

- *tags*

A list of values that specifies to generate a driver and receiver file using the endpoints of a net as a driver based on the type of device pin, h-port, or text port that is attached to the end point of the net. The default tags are c, d, e, and s. The tags you can choose from are

**d**[rain]

**s**[ource]

**t**[ext port] (identifies text ports on any net as a driver)

**c**[ollector]

**e**[mitter]

**neg**

**pos**

#### Description

Instructs the Calibre xRC tool to generate a driver and receiver file in the same way that the Calibre xL tool generates a driver and receiver file when the [PEX Inductance Driver File](#) statement is specified. The default name of the file generated is *driver.xl*, unless you specify a different name or location using the [PEX Inductance Driver File](#) command. The Calibre xL tool calculates self and mutual impedance for all paths that are not filtered.

#### Examples

The following statement instructs the Calibre xRC tool to generate a driver and receiver file using the collector, drain, emitter, and source of nets in the layout as drivers.

```
PEX GENERATE DRIVER FILE TAG c d e s
```

## PEX Generate Driver\_File Tag

Parasitic extraction

**PEX GENERATE DRIVER\_FILE TAG [tags]**

---

**Note**

 PEX Generate Driver\_File Tag has been renamed to [PEX Generate Driver File Tag](#).

There are no changes to parameter definitions and functionality. PEX Generate Driver\_File Tag has been deprecated as of the 2009.3 release.

---

## PEX Ground

Parasitic extraction

**PEX GROUND** [LAYOUT | SOURCE] ***name*** [*name...*]

Used only in Calibre xL

### Parameters

- **LAYOUT**  
An optional keyword that specifies the net names are derived from the layout. This is the default.
- **SOURCE**  
An optional keyword that specifies the net names are derived from the schematic. Ensure the SVRF rule file contains a valid Source Path statement and a valid Source Primary statement identifying the primary cell when specifying this option.
- ***name***  
A required name of a ground net. You can specify ***name*** any number of times in a statement, and the nets are independent of one another. The string ***name*** can contain one or more question mark (?) characters. This wildcard character matches zero or more characters. The ***name*** can be a string variable.

### Description

Specifies one or more ground net names to consider during inductance extraction.

---

#### Note



If the PEX Ground statement is not specified, the Calibre xL tool searches for [LVS](#) [Ground Name](#) statements to identify ground nets. If neither the PEX Ground nor the LVS Ground Name statements are specified, the Calibre xL tool identifies nets with the name VSS? and GND? as ground nets.

---

### Examples

The following statement instructs the Calibre xL tool to use VSS and VSS1 as ground nets.

```
PEX GROUND VSS VSS1
```

# PEX Ground Layer

Parasitic extraction

**PEX GROUND LAYER** *layer\_name* [*layer\_name...*]

## Parameters

- *layer\_name*

A required name of a layer containing the ground shapes. The layer may be original or derived and must not appear in any capacitance rule. Connectivity must be assigned directly.

## Description

Specifies layers containing shapes corresponding to different regions of the substrate to handle multiple grounds. When shapes overlap, the precedence is determined by the layer order, with the overlap being treated as belonging to the layer appearing first.

If you are performing extraction without coupled capacitance using the -c or -cc switch, then the following warning is issued:

WARNING: PEX GROUND LAYER effective only in couple cap extraction.

### Note



The Calibre xACT and Calibre xRC tools handle the PEX Ground Layer statement differently. In Calibre xRC, the ground layer extends virtually so no capacitance escapes around it and the capacitance is coupled to ground. For Calibre xACT, there may be capacitance around the PEX Ground Layer that is coupled to a virtual ground.

## Examples

```
PEX GROUND LAYER analog_regions digital_regions
```

## PEX Ideal Xcell

Parasitic extraction

### PEX IDEAL XCELL {NO | YES}

Used only in Calibre xRC.

#### Parameters

- **NO**  
Keyword that instructs the tool to not write intentional devices from within a cell to the output netlist. NO is the default behavior when the statement is not included.
- **YES**  
Keyword that instructs the tool to write intentional devices in a cell to the output netlist.

#### Description

Specifies whether the Calibre xRC tool should output the intentional devices inside of the xcells to the netlist during the formatting stage. Ideal xcell refers to the intentional devices inside of a cell.

When PEX Ideal Xcell is set to NO and no flags have been specified for any cells in the Xcell list, *all* xcells are treated as primitives and no intentional devices appear in the netlist. If any cells in the Xcell list are specified with the -I flag, the Calibre xRC tool will treat those cells as ideal xcells and the intentional devices for those cells will appear in the netlist.

When PEX Ideal Xcell is set to YES, *all* xcells except for those specified with the -P flag are treated as ideal xcells. Cells specified with the -P flag will be treated as primitives and the contents of those cells do not appear in the netlist. Only gate-level extraction is supported.

PEX Ideal Xcell supports CalibreView, Eldo, HSPICE, DSPF, and Spectre netlists. It does not affect SPEF netlists.

The Calibre xRC tool allows the extraction of devices without parasitics. This is useful for flows where the device models already include the parasitics. For example, when physical layout of devices is done by using device generators or parameterized cells (pcells), and the device models are parameterized to match the layout.

To ensure accurate simulation results when using parameterized models, the extraction tool must not extract parasitics inside the specified devices. Anything in the design's xcell list is treated as ideal, meaning that no parasitics are included, and flattened to the transistor level.

#### Examples

Given the following Xcell file:

```
// Layout_name Source_name flag
NOR    NOR    -I          // treat as an ideal cell
NAND   NAND   -P          // treat as a primitive
INV    INV    // no flag specified
```

**Example 1**

The following statement instructs the Calibre xRC tool to treat the NOR and INV cells as ideal xcells:

```
PEX IDEAL XCELL YES
```

The NAND cell would be treated as a primitive and no intentional devices are written to the netlist for the NAND cell.

**Example 2**

The following statement instructs the Calibre xRC tool to not treat the NAND and INV cells as ideal xcells:

```
PEX IDEAL XCELL NO
```

The NOR cell specified with a -I flag is treated as an ideal xcell and its intentional devices will appear in the netlist.

## PEX Ignore Capacitance

Parasitic extraction

Syntax 1, multi-layer capacitance calculations for all types:

**PEX IGNORE CAPACITANCE [GLOBAL] ALL [SUBSTRATE]**  
*{layer | VIA layer1 layer2} {layer | VIA layer1 layer2} [{layer | VIA layer1 layer2} ...]*

Syntax 2, multi-layer coupled capacitance calculations for non-via layers:

**PEX IGNORE CAPACITANCE [GLOBAL] {FRINGE | NEARBODY | PLATE}**  
*layer1 layer2*  
*[layer ...]*

Syntax 3, intrinsic capacitance calculations for one or more non-via layers:

**PEX IGNORE CAPACITANCE [GLOBAL] INTRINSIC {FRINGE | PLATE} *layer***  
*[layer ...]*

Syntax 4, intrinsic capacitance calculations for all layers:

**PEX IGNORE CAPACITANCE [GLOBAL] INTRINSIC ALL {layer | VIA layer1 layer2}**  
*[{layer | VIA layer1 layer2} ...]*

Syntax 5, complete coupling capacitance calculations for all layers:

**PEX IGNORE CAPACITANCE [GLOBAL] COUPLING [INTRINSIC] {layer | VIA layer1 layer2}**  
*[{layer | VIA layer1 layer2} ...]*

Syntax 6, nearbody capacitance calculations for single-layer:

**PEX IGNORE CAPACITANCE [GLOBAL] NEARBODY {layer | VIA layer1 layer2}**

Syntax 7, intra-device capacitance calculations:

**PEX IGNORE CAPACITANCE [GLOBAL] DEVICE [INTRINSIC] *device\_layer***  
*[device\_layer...] marker\_layer*

Syntax 8, intra-cell capacitance calculations:

**PEX IGNORE CAPACITANCE [GLOBAL] DEVICE [INTRINSIC] *device\_layer***  
*[device\_layer...] CELL *cell\_name* [*cell\_name* ...]*

## Summary

Ignores or removes parasitic capacitances specified through layers and capacitance models ALL, FRINGE, NEARBODY, or PLATE and capacitance types COUPLING, DEVICE, or INTRINSIC.

## Parameters

- Specify one of the following capacitance effects to be ignored, depending on the syntax:
  - ALL** — A required keyword for syntax 1 and syntax 4. Equivalent to specifying the rule independently for NEARBODY, FRINGE, and PLATE together.
  - FRINGE** — A required keyword for syntax 2 and syntax 3. Ignores the capacitance between the side of a wire and either the substrate or the bottom or top of a wire.

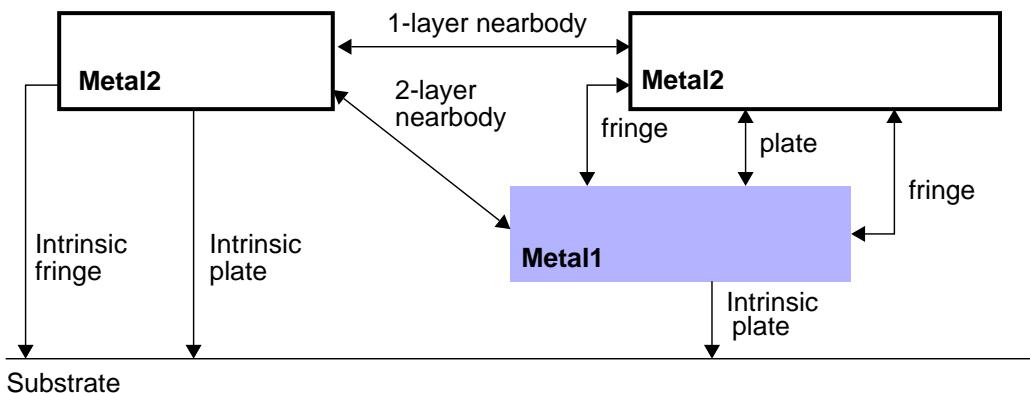
**NEARBODY** — A required keyword for syntax 2 and syntax 6. Ignores the capacitance between the sides of two wires, either on the same layer or different layers.

**PLATE** — A required keyword for syntax 2 and syntax 3. Ignores the capacitance between the lower surface of a wire and substrate, or the lower surface of a wire to the upper surface of another wire.

For a pictorial definitions of the different effects, see [Figure 4-276](#) and [Figure 4-277](#).

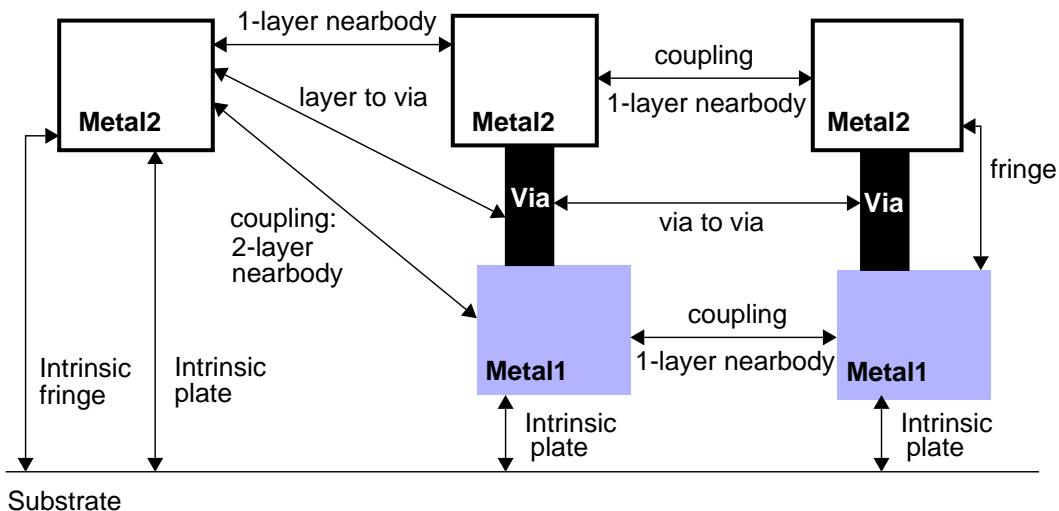
**Figure 4-276. Definition of Capacitance Types**

Cross-section view



**Figure 4-277. Definition of Capacitance Types with Vias**

Cross-section view



- Specify one of the following, depending on the syntax:

**SUBSTRATE** — An optional keyword for syntax 1. Specifies that all intrinsic capacitance for all layers specified will be ignored.

**COUPLING** — A required keyword only used in syntax 5. Specifies that all coupling capacitances for all layers will be ignored.

**INTRINSIC** — A required keyword for syntax 3 and syntax 4. An optional keyword for syntax 5, syntax 7, and syntax 8. For syntax 3, it is only valid with FRINGE and PLATE capacitance effects. Syntax 4 specifies that all intrinsic capacitances will also be ignored. For syntax 5, it specifies that intrinsic capacitances will also be ignored. For syntax 7 and syntax 8, it specifies that intrinsic capacitance will be ignored for the specified device layers.

**CELL** — A required keyword for syntax 8. Specifies that device capacitances in named cells will be ignored.

**DEVICE** — A required keyword for syntax 7 and syntax 8. Specifies that device capacitances in named cells or between layers identified by touching or abutting the **marker\_layer** will be ignored.

- **GLOBAL**

An optional keyword used to apply the specified ignore to all physical layers mapped to the same logical layer. This option is only valid with layers that have been specified in a **PEX Map** statement. It may not be used with layers specified in a **PEX Alias** statement.

- **layer**

Specifies an original or derived non-via layer.

- **VIA layer1 layer2**

Keyword and parameter set that specifies a via defined between **layer1** and **layer2**.

- **device\_layer**

Specifies an original or derived layer, used to define a device or a layer in a device.

- **marker\_layer**

Specifies an original or derived layer, used to mark the extent of a device.

- **cell\_name**

Specifies a device that is defined by the contents of one more cells.

## Description

Specifies to ignore or remove all capacitance from the particular interaction given by one of the capacitance effects, ALL, FRINGE, NEARBODY, or PLATE for the layer or layers listed. All layers must be properly defined. Interconnect and via layers can be specified. This statement cannot be used with **PEX Elayer**.

**Note**

If you ignore a capacitance effect for a layer defined as a *primary\_layer* in PEX Alias, all *sublayers* also ignore the capacitance effect. This is universally true, even if the sublayers have a sublayer-specific rule for the capacitance effect defined.

## Using Syntax 1

For syntax 1, all of the capacitance calculations between any two layers will be ignored. This also includes via layers. Specifying a layer name twice in a single statement causes the coupling capacitance for that layer to be ignored.

The SUBSTRATE option can be added to ignore all intrinsic capacitance for the specified layers as well.

### Syntax 1 Example 1

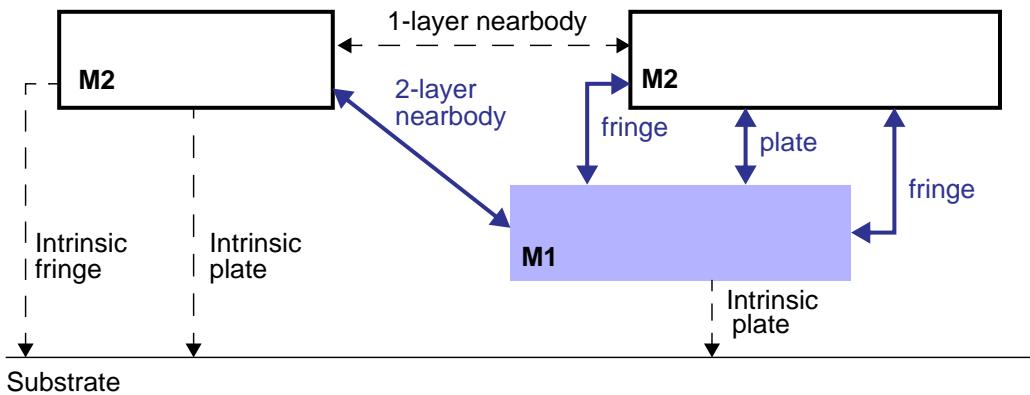
The statement

```
PEX IGNORE CAPACITANCE ALL M1 M2
```

will ignore the capacitance calculations between the layers M1 and M2 for nearbody, fringe, and plate effects. [Figure 4-278](#) shows an example of the effects that are ignored.

**Figure 4-278. Ignored Capacitance Types for Syntax 1 Example 1**

#### Cross-section view



### Syntax 1 Example 2

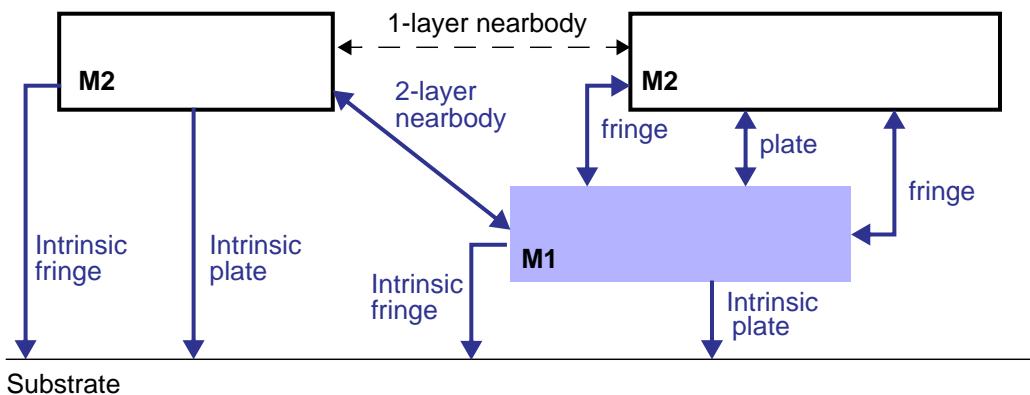
The statement

```
PEX IGNORE CAPACITANCE ALL SUBSTRATE M1 M2
```

will ignore the capacitance calculations between the layers M1 and M2 for nearbody, fringe, and plate effects, and all intrinsic capacitance for layers M1 and M2. [Figure 4-279](#) shows an example of the effects that are ignored.

**Figure 4-279. Ignored Capacitance Types for Syntax 1 Example 2**

Cross-section view

**Syntax 1 Example 3**

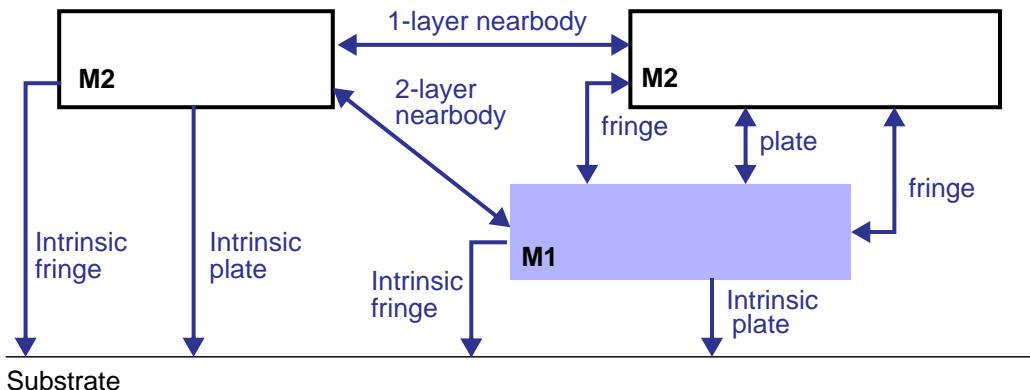
The statement

```
PEX IGNORE CAPACITANCE ALL SUBSTRATE M1 M2 M2
```

will ignore all capacitance calculations between the layers M1 and M2, and all 1-layer nearbody capacitance for layer M2. [Figure 4-280](#) shows an example of the effects that are ignored.

**Figure 4-280. Ignored Capacitance Types for Syntax 1 Example 3**

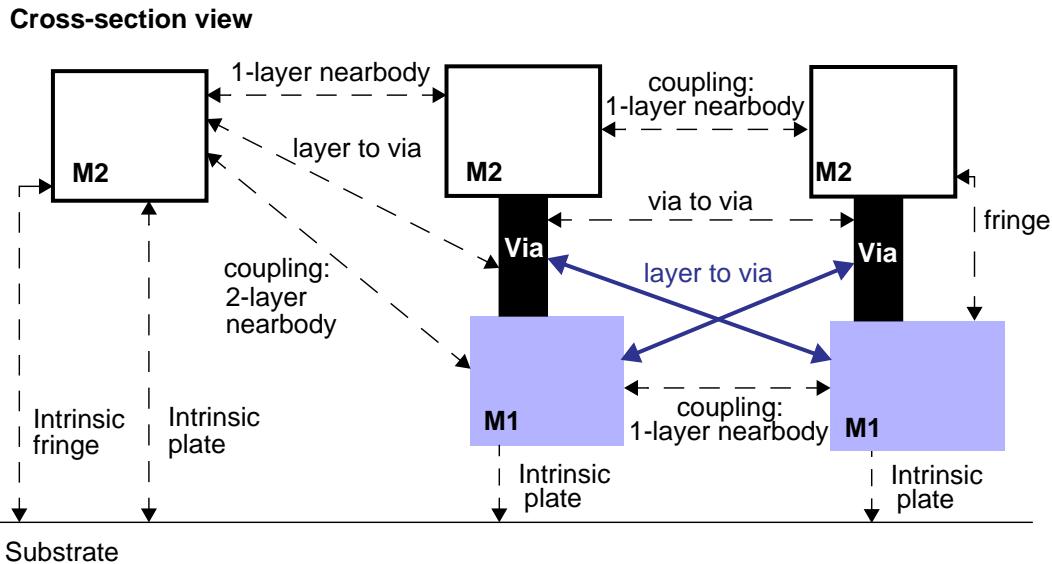
Cross-section view

**Syntax 1 Example 4**

The statement

```
PEX IGNORE CAPACITANCE ALL M1 VIA M1 M2
```

will ignore the capacitance calculations between layer M1 and any vias between layers M1 and M2. [Figure 4-281](#) highlights the effects that are ignored.

**Figure 4-281. Ignored Capacitance Types with Vias for Syntax 1 Example 4**

## Using Syntax 2

For syntax 2, the indicated capacitance calculations between any two layers will be ignored regardless of the order of the layers; that is, both the rules for *layer1-layer2* and *layer2-layer1* effects are not run. You cannot ignore the capacitance rules for effects in only one direction.

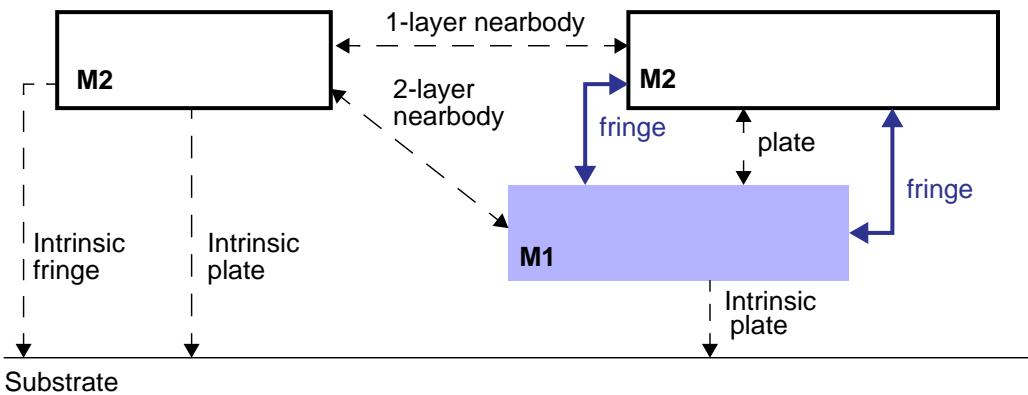
When more than two layers are specified, all permutations of the layers are ignored. This syntax applies to non-via layers only.

### Syntax 2 Example

The statement

```
PEX IGNORE CAPACITANCE FRINGE M1 M2
```

will ignore the capacitance calculations between the layers M1 and M2 for fringe effects. [Figure 4-282](#) shows an example of the effects that are ignored.

**Figure 4-282. Ignored Capacitance Types for Syntax 2 Example****Cross-section view****Using Syntaxes 3 and 4**

Intrinsic capacitance is a one-layer effect. It represents the capacitance between the named layer and substrate. Specifying multiple layers in a single statement is the same as specifying an individual PEX Ignore Capacitance INTRINSIC statement for each layer.

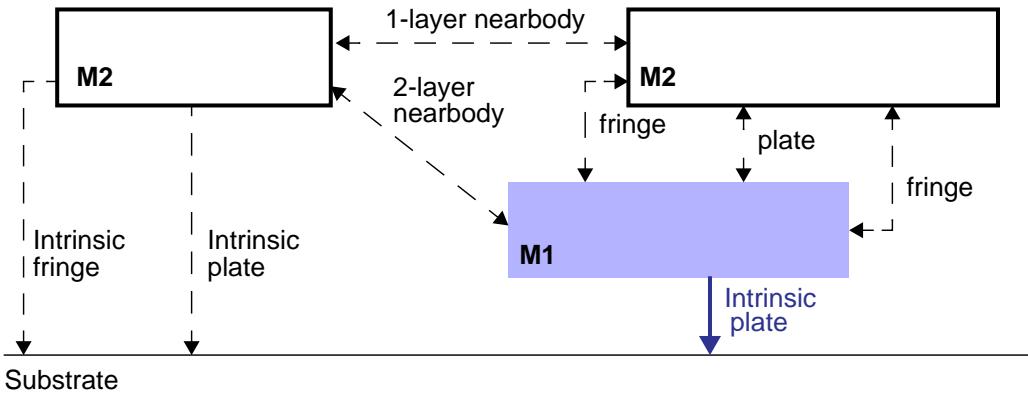
Syntax 3 ignores calculation for FRINGE and PLATE intrinsic capacitances only. Use this syntax for non-via layers.

**Syntax 3 Example**

The statement

```
PEX IGNORE CAPACITANCE INTRINSIC PLATE M1
```

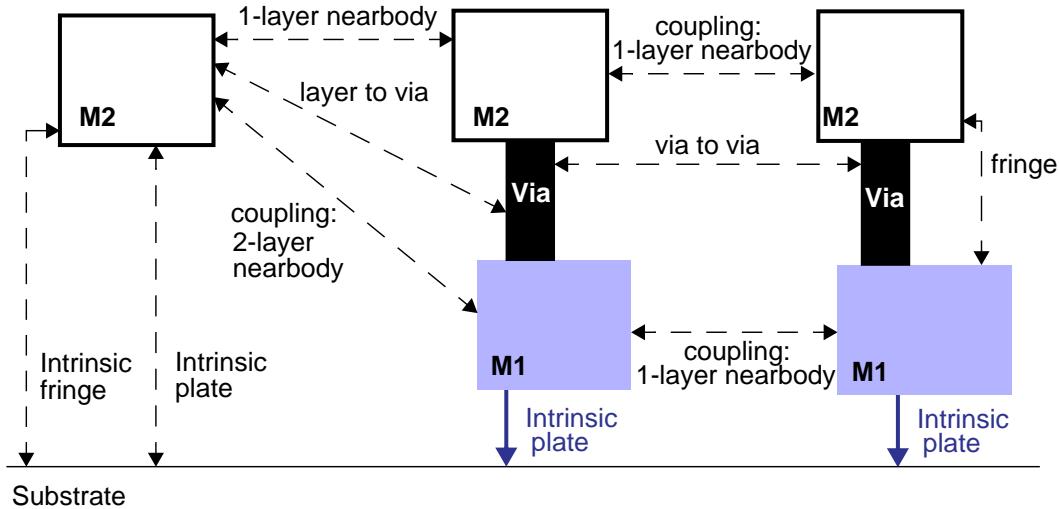
will ignore the capacitance calculations between the layers M1 and substrate for intrinsic plate effects. [Figure 4-283](#) and [Figure 4-284](#) show examples of the effects that are ignored.

**Figure 4-283. Ignored Capacitance Types for Syntax 3 Example****Cross-section view**

Note that in [Figure 4-284](#) the intrinsic via capacitance is not included.

**Figure 4-284. Ignored Capacitance Types with Vias for Syntax 3**

**Cross-section view**



Syntax 4 ignores all calculations for all layers, including via layers.

**Syntax 4 Example 1**

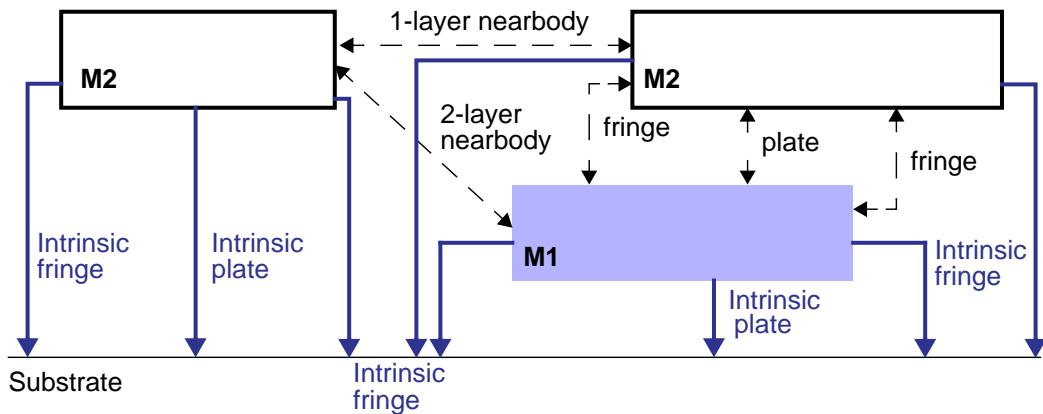
The statement

```
PEX IGNORE CAPACITANCE INTRINSIC ALL M1 M2
```

will ignore the capacitance calculations between the layer M1 and substrate and layer M2 and substrate for all intrinsic capacitance effects (both plate and fringe). [Figure 4-285](#) show examples of the intrinsic effects that are ignored.

**Figure 4-285. Ignored Capacitance Types for Syntax 4 Example 1**

**Cross-section view**



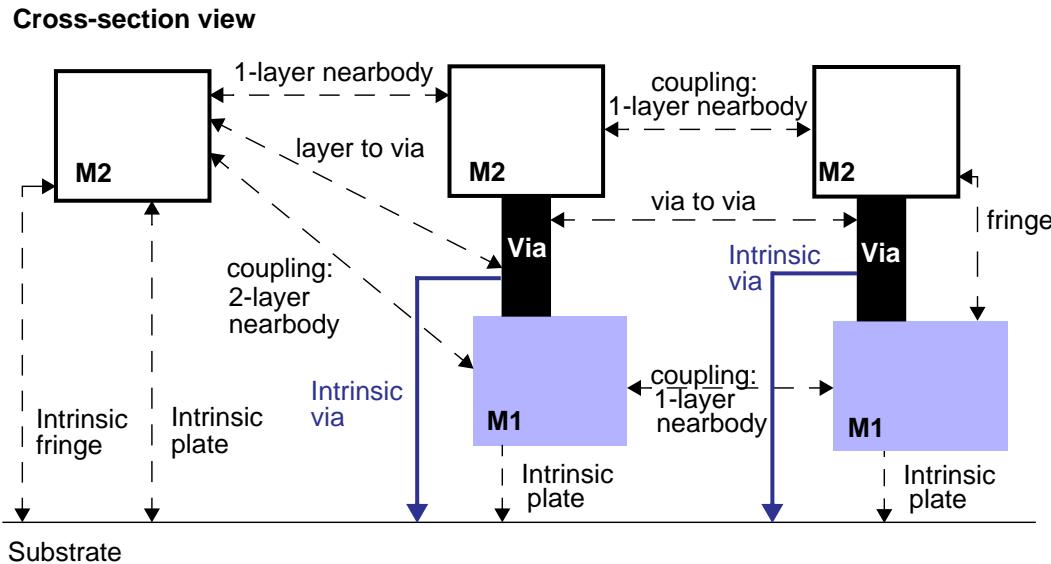
**Syntax 4 Example 2**

The statement

```
PEX IGNORE CAPACITANCE INTRINSIC ALL VIA M1 M2
```

will ignore the intrinsic capacitance for vias between layers M1 and M2. [Figure 4-286](#) highlights the effects that are ignored.

**Figure 4-286. Ignored Capacitance Types with Vias for Syntax 4 Example**

**Using Syntax 5**

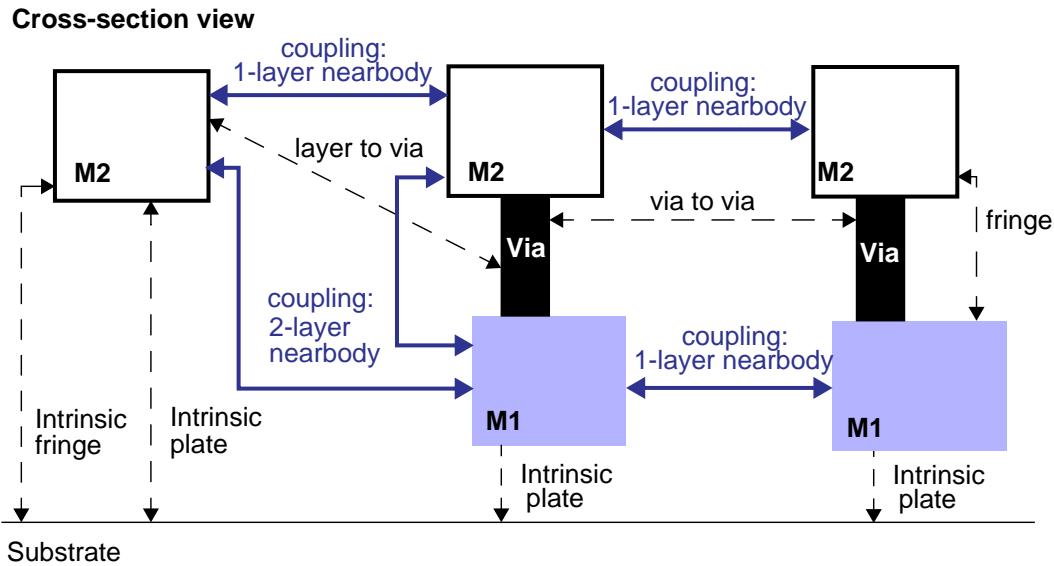
For syntax 5, all coupling capacitance between listed layers will be ignored. Coupled capacitance to other layers will still be extracted.

**Syntax 5 Example**

The statement

```
PEX IGNORE CAPACITANCE COUPLING M1 M2
```

will ignore the coupling capacitance between the M1 layers, the M2 layers, and between the M1 and M2 layers as well. [Figure 4-287](#) highlights the effects that are ignored.

**Figure 4-287. Ignored Capacitance Types with Vias for Syntax 5 Example**

## Using Syntax 6

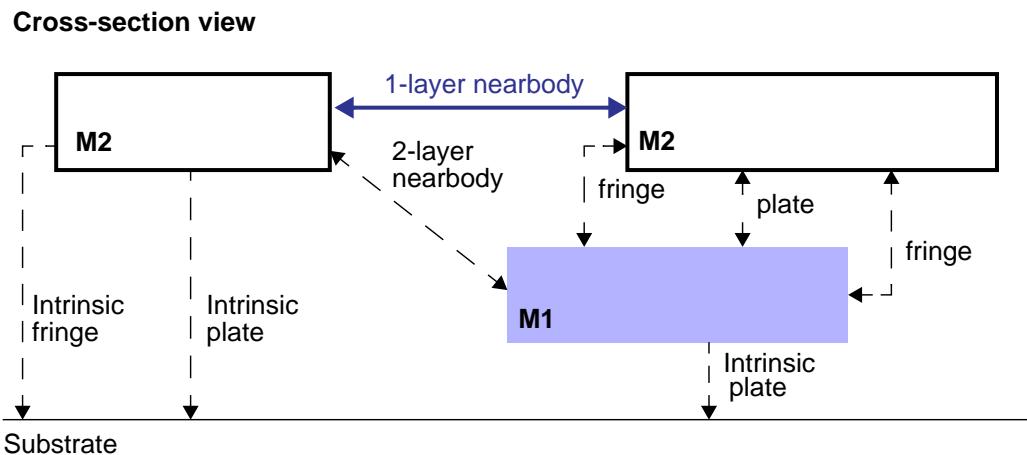
For syntax 6, nearbody capacitance calculations are ignored between polygons on the same layer. You can specify either via or non-via layers. Only a single layer or via layer pair may be specified.

### Syntax 6 Example 1

The statement

```
PEX IGNORE CAPACITANCE NEARBODY M2
```

will ignore the nearbody capacitance between the geometries on layer M2. [Figure 4-288](#) shows an example of the effects that are ignored.

**Figure 4-288. Ignored Capacitance Types for Syntax 6 Example 1**

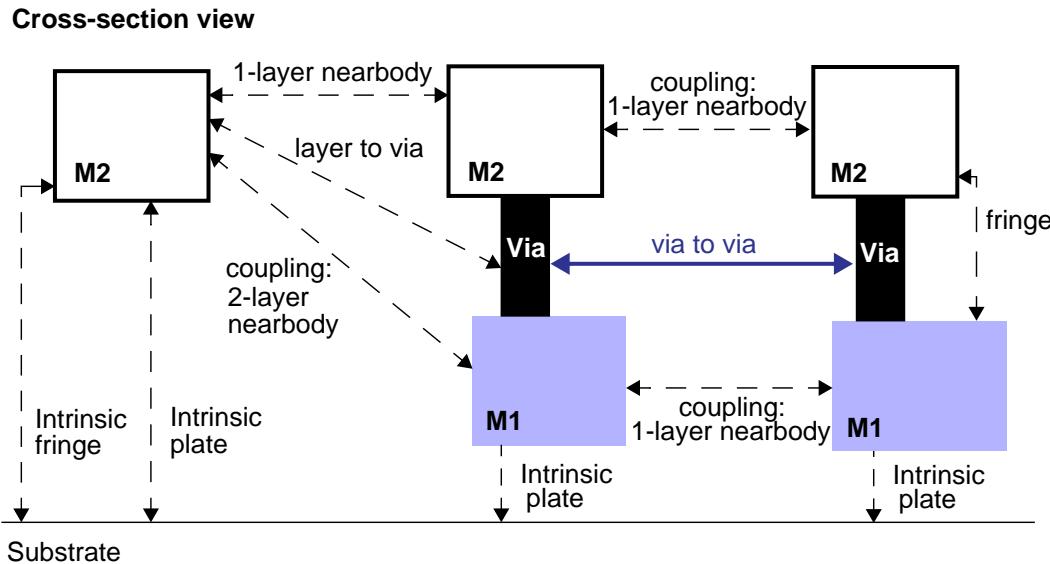
**Syntax 6 Example 2**

The statement

```
PEX IGNORE CAPACITANCE NEARBODY VIA M1 M2
```

will ignore the capacitance calculations for vias between layers M1 and M2. [Figure 4-289](#) highlights the effects that are ignored.

**Figure 4-289. Ignored Capacitance Types with Vias for Syntax 6 Example 2**

**Using Syntaxes 7 and 8**

Syntaxes 7 and 8 can be used to allow the calculation of parasitic capacitance between devices while excluding capacitance for layers inside of a device.

Syntax 7 prevents extraction of capacitance between device layers that are touching the **marker\_layer** polygon. For example, metal1-metal2 parallel plates in a device capacitor that are touching the **marker\_layer** are excluded from capacitance extraction. However, if many instances of the same type of capacitor are placed close to each other, the parasitic capacitance between these is calculated.

Use syntax 8 when a device is defined by the contents of one or more cells.

**Examples**

The statement

```
PEX IGNORE CAPACITANCE ALL M1 M2 M3
```

is equivalent to including the following statements:

```
PEX IGNORE CAPACITANCE PLATE M1 M2
```

```
PEX IGNORE CAPACITANCE PLATE M1 M3  
PEX IGNORE CAPACITANCE PLATE M2 M3  
  
PEX IGNORE CAPACITANCE FRINGE M1 M2  
PEX IGNORE CAPACITANCE FRINGE M1 M3  
PEX IGNORE CAPACITANCE FRINGE M2 M3  
  
PEX IGNORE CAPACITANCE NEARBODY M1 M2  
PEX IGNORE CAPACITANCE NEARBODY M1 M3  
PEX IGNORE CAPACITANCE NEARBODY M2 M3
```

It does not include these rules:

```
PEX IGNORE CAPACITANCE NEARBODY M1  
PEX IGNORE CAPACITANCE NEARBODY M2  
PEX IGNORE CAPACITANCE NEARBODY M3  
PEX IGNORE CAPACITANCE INTRINSIC ALL M1  
PEX IGNORE CAPACITANCE INTRINSIC ALL M2  
PEX IGNORE CAPACITANCE INTRINSIC ALL M3
```

## PEX Ignore Inductance

Parasitic extraction

**PEX IGNORE INDUCTANCE [GLOBAL] *layer* [*layer...*]**

### Parameters

- GLOBAL

An optional keyword used to apply the specified ignore to all physical layers mapped to the same logical layer. This option is only valid with layers that have been specified in a [PEX Map](#) statement. This option may not be used with layers specified in a [PEX Alias](#) statement.

- *layer*

One or more original or derived layers. At least one layer must be specified.

### Description

Ignores any inductance for the specified layer or connections during extraction. Inductance is not computed for any segments that belong to the ignored layer. Connectivity is unchanged but the section or cell contributes no inductance to the parasitic extraction results.

This command cannot be used with [PEX Elayer](#).

### Examples

To ignore inductance from metal1 layer specify the following statement in the rule file:

```
PEX IGNORE INDUCTANCE metal1
```

# PEX Ignore Resistance

Parasitic extraction

**PEX IGNORE RESISTANCE [GLOBAL] *layer* [*layer*...]**

**PEX IGNORE RESISTANCE [GLOBAL] INDIE *layer* [*layer*...]**

**PEX IGNORE RESISTANCE [GLOBAL] VIA *layer1* *layer2* [VIA *layer1* *layer2*]...**

**PEX IGNORE RESISTANCE [GLOBAL] DEVICE *layer* [*layer* ...] [VIA *layer1* *layer2* ... ]  
CELL *cell\_name* [*cell\_name* ...]**

## Parameters

- **GLOBAL**

An optional keyword used to apply the specified ignore to all physical layers mapped to the same logical layer. This option is only valid with layers that have been specified in a [PEX Map](#) statement. This option may not be used with layers specified in a [PEX Alias](#) statement.

- ***layer***

One or more original or derived layers. At least one layer must be specified.

- **INDIE**

Specifies the fill layers that should not have in-die variation applied when calculating resistance.

- **VIA *layer1* *layer2***

Required keyword set that specifies a via by defining its connection layers. You can specify more than one via connection.

- **DEVICE**

Specifies that via resistances in named cells will be ignored.

- **CELL *cell\_name***

Required keyword set that specifies that via resistances are ignored in named cells. You can specify more than one cell.

## Description

Ignores any resistivity for the specified layer or connections during extraction. Connectivity is unchanged but the section or cell contributes no resistance to the parasitic extraction results.

If the rule file specifies PEX Ignore Resistance VIA for two layers, the resistances of any vias between them are automatically ignored.

If the rule file specifies PEX Ignore Resistance DEVICE VIA ... CELL, the resistance of any vias inside the named cells are automatically ignored.

If the rule file specifies PEX Ignore Resistance INDIE, then ignore in-die variation when calculating resistance for the specified layers. Resistance calculations for the specified layers will be based on drawn values.

## **PEX Ignore Resistance**

---

This command cannot be used with [PEX Elayer](#).

### **Examples**

To ignore resistance from metal 1 and connection resistances between metal1 and diffusion layers in the nand1 and nor1 cells, specify the following statement in the rule file:

```
PEX IGNORE RESISTANCE DEVICE metall VIA tndiff metall VIA tpdiff metall  
CELL nand1 nor1
```

# PEX Include Distributed

Parasitic extraction

**Note**



As of the 2008.4 release, this statement has been deprecated. Use [PEX Extract Include](#) instead.

## PEX INCLUDE DISTRIBUTED [LAYOUT | SOURCE] [TOPLEVEL | RECURSIVE]

*name* [*name* ...]

Used only in Calibre xRC.

### Parameters

- LAYOUT

Optional keyword that specifies the names are derived from the layout. This is the default behavior if you do not include a choice from this set in the statement.

- SOURCE

Optional keyword that specifies the names are derived from the source.

- TOPLEVEL

Specifies that the search is to take place in the top level. This is the default behavior if you do not include a choice from this set.

- RECURSIVE

Specifies that the search is to take place on all levels.

- ***name***

A required net to be included for RC extraction. You can specify ***name*** any number of times in one statement. The ***name*** can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Names that include wildcards must be enclosed in quotation marks.

### Description

Specifies to limit distributed RC extraction to the listed net names. That is, only specified nets are processed.

Multiple statements form a single combined set.

The net names can include the question mark character (?) as a wildcard. The tool supports wildcards in the net name only, not in the path. For example, “*X0/X1/foo?*” is supported, but “*?/X0/vdd*” is not supported. In other words, the wildcard character does not match hierarchy. Names that include wildcards must be enclosed in quotation marks.

The rule file compiler will fail if you attempt to use the TOPLEVEL keyword with wildcards in the path. The rule file compiler will also fail if you use the RECURSIVE keyword with pathnames used to specify a specific net.

## Examples

```
PEX INCLUDE DISTRIBUTED in1 net1
PEX INCLUDE DISTRIBUTED SOURCE RECURSIVE "?in?" "?out?"
```

# PEX Include Lumped

Parasitic extraction

**Note**



As of the 2008.4 release, this statement has been deprecated. Use [PEX Extract Include](#) instead.

## PEX INCLUDE LUMPED [LAYOUT | SOURCE] [TOPLEVEL | RECURSIVE]

*name* [*name* ...]

Used only in Calibre xRC.

### Parameters

- LAYOUT

Optional keyword that specifies the names are derived from the layout. This is the default behavior if you do not include a choice from this set in the statement.

- SOURCE

Optional keyword that specifies the names are derived from the source.

- TOPLEVEL

Specifies that the search is to take place in the top level. This is the default behavior if you do not include a choice from this set.

- RECURSIVE

Specifies that the search is to take place on all levels.

- ***name***

A required net to be included for lumped C extraction. You can specify ***name*** any number of times in one statement. The ***name*** can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. Names that include wildcards must be enclosed in quotation marks.

### Description

Specifies to limit lumped C extraction to the listed net names. That is, only specified nets are processed.

Multiple statements form a single combined set.

The net names can include the question mark character (?) as a wildcard. The tool supports wildcards in the net name only, not in the path. For example, “*X0/X1/foo?*” is supported, but “*?/X0/vdd*” is not supported. In other words, the wildcard character does not match hierarchy. Names that include wildcards must be enclosed in quotation marks.

The rule file compiler will fail if you attempt to use the TOPLEVEL keyword with wildcards in the path. The rule file compiler will also fail if you use the RECURSIVE keyword with pathnames used to specify a specific net.

### Examples

```
PEX INCLUDE LUMPED in1 net1
PEX INCLUDE LUMPED SOURCE RECURSIVE "?in?" "?out?"
```

# PEX Indie Spacing

Parasitic extraction

## PEX INDIE SPACING {EDGE BASED | AVERAGE}

### Parameters

- **EDGE BASED**

Use the spacing for the side of the polygon to calculate the bias for its edge.

- **AVERAGE**

Use a single spacing value to calculate the bias for both edges of the polygon.

### Description

Defines a way to calculate bias for a polygon as a function of its spacing. This is a foundry pre-defined statement.

For more information on in-die variation, see “[Handling Parasitic On-Chip Variation](#)” in the *Calibre xRC User’s Manual*.

## PEX Inductance Default Partial Model

Parasitic extraction

### PEX INDUCTANCE DEFAULT PARTIAL MODEL {ON | OFF}

Used only in Calibre xL.

#### Parameters

- **ON**

A required keyword that specifies to use PI. This is the default if the statement is not specified.

- **OFF**

A required keyword that specifies to use PUL.

#### Description

Specifies whether to use partial inductance (PI) or per unit length (PUL) when calculating self impedance for paths that are 2 microns or longer and have no return path near them. If this statement is not used in your SVRF rule file or the argument is not specified, self and mutual impedance is calculated using PI rather than PUL.

#### Examples

The following statement instructs the Calibre xL tool to use PUL when calculating the self impedance for paths that are 2 microns or longer and have no return path near them.

```
PEX INDUCTANCE DEFAULT PARTIAL MODEL OFF
```

## PEX Inductance Default PI

Parasitic extraction

### PEX INDUCTANCE DEFAULT PI{ON | OFF}

---

**Note**

 PEX Inductance Default PI has been renamed to [PEX Inductance Default Partial Model](#).

There are no changes to parameter definitions and functionality. PEX Inductance Default PI has been deprecated as of the 2009.1 release.

---

## PEX Inductance Differential Pair

Parasitic extraction

---

**Note**

As of the 2010.2 release, this statement has been deprecated. Use [PEX Inductance Victim](#) instead to specify the pairs of nets targeted for mutual inductance extraction. For more information on differential pair extraction using the standard mutual inductance flow, see “[Differential Pair Extraction](#)” in the [Calibre xL User’s Manual](#).

---

### PEX INDUCTANCE DIFFERENTIAL PAIR [LAYOUTNAMES | SOURCENAMES]

{*net\_name\_1* *net\_name\_2* [*return\_net\_name*]}

Used only in Calibre xL.

#### Parameters

- LAYOUTNAMES

An optional keyword that specifies the net names are derived from the layout. This is the default behavior if this statement does not appear in your rule file.

- SOURCENAMES

An optional keyword that specifies the net names are derived from the schematic. Ensure the SVRF rule file contains a valid Source Path statement and a valid Source Primary statement identifying the primary cell when specifying this option.

- *net\_name\_1*

A required name identifying a net in the differential pair.

- *net\_name\_2*

A required name identifying another net in the differential pair.

- *return\_net\_name*

An optional name identifying the return net for the differential pair.

#### Description

Specifies the two or more nets that are part of the differential pair. You can specify a differential pair or a group of differential pairs in one or more statements. You can also identify these nets using the Calibre RVE tool.

This statement generates partial self inductance and mutual inductance for the listed differential pair nets. Only use this statement for differential pairs that operate and will be simulated in differential mode. Using a general loop-based solution is highly recommended for extracting accurate differential pair impedance and subsequent simulation.

**Note**

 Using this statement, you can optionally specify a third net. The third net is considered as the local return path of the differential pair. This feature is not supported when selecting differential pairs using the Calibre RVE tool.

---

## Examples

The following examples instruct the Calibre xL tool to calculate self impedance for all nets and mutual impedance between the different combinations of differential pair nets.

### Example 1

The following statement instructs the Calibre xL tool to calculate mutual impedance between nets IN\_P and IN\_M.

```
PEX INDUCTANCE DIFFERENTIAL PAIR SOURCENAMES IN_P IN_M
```

### Example 2

Instructs the Calibre xL tool to calculate mutual impedance between the differential pair nets only; that is, between nets P6 and P11, P9 and P7, and P18 and P19.

```
PEX INDUCTANCE DIFFERENTIAL PAIR P6 P11  
PEX INDUCTANCE DIFFERENTIAL PAIR P9 P7  
PEX INDUCTANCE DIFFERENTIAL PAIR P18 P19
```

## PEX Inductance Doprocess

Parasitic extraction

**PEX INDUCTANCE DOPROCESS** *layer\_name* [*layer\_name...*]

---

**Note**

 PEX Inductance Doprocess has been renamed to [PEX Inductance Extract Layers](#). There are no changes to parameter definitions and functionality. PEX Inductance Doprocess has been deprecated as of the 2009.1 release.

---

# PEX Inductance Driver File

Parasitic extraction

## PEX INDUCTANCE DRIVER FILE *driver\_and\_receiver\_file\_location*

Used only in Calibre xL.

### Parameters

- *driver\_and\_receiver\_file\_location*

A required parameter that specifies the location of the driver and receiver file.

### Description

Specifies the location of the driver and receiver file. The default is “*./driver.xl*”. You need to specify this statement if you use a different file name or the file resides in a location other than your working directory.

### Examples

The following statement instructs the tool to search for a driver and receiver file named *driver.xl* in the user directory.

```
PEX INDUCTANCE DRIVER FILE "user/driver.xl"
```

## PEX Inductance Driver Summary

Parasitic extraction

### PEX INDUCTANCE DRIVER SUMMARY {ON | OFF}

Used only in Calibre xL.

#### Parameters

- **ON**  
A required keyword which instructs Calibre xL to output a summary of all the nets and their associated driver ports to the transcript.
- **OFF**  
A required keyword that suppresses the report from the transcript. This is the default behavior.

#### Description

This statement allows the user to control the output of the driver summary data to the transcript. The default setting OFF as this information may only be useful for debugging a particular path or when using a driver receiver file for filtering. You can specify this statement once in your SVRF rule file.

#### Examples

To output the layer summary to the transcript, use the following:

```
PEX INDUCTANCE DRIVER SUMMARY ON
```

The resulting driver summary will look like:

```
-----  
DRIVER SUMMARY:  
  
Net      Driver  
  
VCTRL    "VCTRL T"  
15       "X2 MINUS"  
16       "X0 PLUS"  
17       "X0 MINUS"  
18       "X25/X0 MINUS"  
19       "X24/X0 MINUS"  
20       "X20/X0 MINUS"  
21       "X21/X0 MINUS"  
VGS      "VGS T"  
25       "X1 D"  
VOUT_L   "VOUT_L T"  
VOUT_R   "VOUT_R T"
```

# PEX Inductance Extract Layers

Parasitic extraction

**PEX INDUCTANCE EXTRACT LAYERS** *layer\_name* [*layer\_name...*]

Used only in Calibre xL.

## Parameters

- *layer\_name*

A required argument identifying a layout layer name. It can be specified any number of times.

---

### Note



If you omit a layout layer from this statement, the Calibre xL tool does not process the layer for inductance extraction.

---

## Description

This optional statement declares the layers to process during inductance extraction. You can specify this statement once in your SVRF rule file.

## Examples

The following statement instructs the Calibre xL tool to process only geometries on met1, met2, and met3 layers during inductance extraction.

```
PEX INDUCTANCE EXTRACT LAYERS met1 met2 met3
```

## PEX Inductance Filter

Parasitic extraction

**PEX INDUCTANCE FILTER {APPROX | OFF | {FILE [TAG *tag* ... ]} | EXCLUSIVE}**

Used only in Calibre xL.

### Summary

Specifies whether or not to filter paths for inductance extraction.

### Parameters

- **APPROX**

A required keyword that specifies to filter nets by approximating one end point as a driver for each net using the default driver tags and extract inductance for the paths from that driver only if they meet the length and frequency criteria defined within the Calibre xL tool. A driver and receiver file is not used. This is the default if you do not specify this statement.

- **OFF**

A required keyword that specifies to not filter nets. Inductance is extracted for all the paths.

- **FILE**

A required keyword that causes the Calibre xL tool to use the drivers and receivers from a file when filtering. The FILE keyword is used to accurately recognize the current paths for each net by listing the driver and receivers for that net. Once the path is recognized, it determines if the path meets the filtering criteria or if it should be filtered out. Any net that does not have any driver or receiver entries in the file will have its current paths recognized using the same method that is applied when the “APPROX” option is specified.

By default, the tool will search for a file named *driver.xl*, unless you specify a different name or location using the **PEX Inductance Driver File** command. The Calibre xL tool calculates self and mutual impedance for all paths that are not filtered.

- **TAG *tag***

An optional keyword followed by a list of values that specifies to generate a driver and receiver file using the end points of a net as a driver based on the type of device pin, h-port, or text port that is attached to the end point of the net. This keyword must be specified with the FILE keyword. The default tags are c, d, e, and s. Some of the commonly used tags you can choose from are

**d**[rain]

**s**[ource]

**t**[ext port] (identifies text ports on any net as a driver)

**c**[ollector]

**e**[mitter]

**neg**

**pos**

- **EXCLUSIVE**

A required keyword used to enable exclusive path selection. Similar to the FILE keyword, EXCLUSIVE causes the Calibre xL tool to use the drivers and receivers from a file when filtering. The EXCLUSIVE keyword is used to accurately recognize the current paths for each net by listing the driver and receiver for that net. Once the path is recognized, it determines if the path meets the filtering criteria or if it should be filtered out. Any net that does not have any driver or receiver entries in the file will have a default inductance value assigned.

By default, the tool will search for a file named “*./driver.xl*.”, unless you specify a different name or location using the [PEX Inductance Driver File](#) statement. For more information on creating a driver file, see “[Generating the Driver and Receiver File](#)” in the [Calibre xL User’s Manual](#).

**Note**

For each wire part of a path filtered by the statement, a default inductance value is specified. This default value is specified using the [PEX Inductance Self](#) statement.

**Description**

Specifies whether or not to filter paths for inductance extraction. The Calibre xL tool does not calculate self or mutual impedance on filtered paths. However, filtered paths are assigned a default self inductance, which you can specify using the [PEX Inductance Self](#) statement.

**Examples****Example 1**

The following statement instructs the Calibre xL tool to generate a driver and receiver file using collector, drain, emitter, and source for each net in the layout as drivers:

```
PEX INDUCTANCE FILTER FILE TAG c d e s
```

The tool will use the generated file in recognizing the current paths and then filter out the paths that don’t meet the length and frequency criteria.

**Example 2**

The following statement instructs the Calibre xL tool to approximate the current paths and then filter out the paths that don’t meet the length and frequency criteria:

```
PEX INDUCTANCE FILTER APPROX
```

### Example 3

The following statement instructs the Calibre xL tool to extract inductance for all nets regardless of length or frequency:

```
PEX INDUCTANCE FILTER OFF
```

### Example 4

The following statement instructs the Calibre xL tool to use the provided driver and receiver file in recognizing the current paths, and then filter out the paths that don't meet the length and frequency criteria:

```
PEX INDUCTANCE FILTER FILE
```

## PEX Inductance Forward Coupling

Parasitic extraction

### PEX INDUCTANCE FORWARD COUPLING {YES | NO}

Used only in Calibre xL.

#### Parameters

- **YES**  
Turns on forward coupling correction. This is the default behavior.
- **NO**  
Turns off forward coupling correction.

#### Description

Specifies that the Calibre xL tool turns on or off forward coupling calculations for self impedance. Turning off forward coupling corrections improves performance at the expense of losing accuracy. You can specify this statement once in your SVRF rule file.

For digital ASIC designs, it is recommended that you disable forward coupling correction. To do this, use the **NO** parameter to improve performance. Specify extracting selected nets using the -select command line parameter during the parasitic extraction stage (-pdb -select).

#### Examples

To turn off forward coupling calculations, use the following:

```
PEX INDUCTANCE FORWARD COUPLING NO
```

## PEX Inductance Frequency

Parasitic extraction

### PEX INDUCTANCE [MAXIMUM] FREQUENCY *hertz*

Used only in Calibre xL.

#### Parameters

- MAXIMUM

An optional keyword that specifies *hertz* is the global maximum operating frequency, not the global operating frequency. If the MAXIMUM keyword is used, the Calibre xL tool uses the broadband model.

- *hertz*

A required parameter that specifies the frequency in hertz. The format is a floating-point number expressed in scientific notation. The default is 1.0e10 (10 GHz) if this statement is not specified.

#### Description

Specifies the global operating frequency or the global maximum operating frequency for all the signal nets in the design. It affects how the inductance of a small metal segment is calculated and how long it is compared to the wavelength. You can override this global value for specific nets using the [PEX Inductance ... Frequency](#) statement.

---

#### Note



If both the PEX Inductance Frequency and [PEX Inductance Switch Time](#) statements are specified in the rule file, the Calibre xL tool uses the broadband model and the highest frequency specified in the statements.

---

#### Examples

##### Example 1

The following statement instructs the Calibre xL tool to use 4 GHz as the global operating frequency for all signal nets in the design during inductance extraction.

```
PEX INDUCTANCE FREQUENCY 4e9
```

##### Example 2

The following statement instructs the Calibre xL tool to use 4 GHz as the global maximum operating frequency for all signal nets in the design during inductance extraction.

```
PEX INDUCTANCE MAXIMUM FREQUENCY 4e9
```

# PEX Inductance ... Frequency

Parasitic extraction

**PEX INDUCTANCE** [LAYOUTNAMES | SOURCENAMES] *net\_name* [MAXIMUM]  
FREQUENCY *hertz*

## Parameters

- **LAYOUTNAMES**  
An optional keyword that specifies the net names are derived from the layout.
- **SOURCENAMES**  
An optional keyword that specifies the net names are derived from the schematic. When specifying SOURCENAMES, ensure the SVRF rule file contains a valid Source Path statement and a valid Source Primary statement identifying the primary cell.
- ***net\_name***  
A required argument specifying the name of the net to be used for analysis.
- **MAXIMUM**  
An optional keyword that specifies *hertz* is the maximum operating frequency of the specified net, not the operating frequency. If the MAXIMUM keyword is used, the Calibre xL tool uses the broadband model for the specified net.
- ***hertz***  
A required keyword and value that specifies the operating frequency of the specified net. The format is a floating-point number in scientific notation.

## Description

Specifies the nets for which the operating frequency or maximum operating frequency is different from the global (maximum) operating frequency specified with the [PEX Inductance Frequency](#) statement. You can have multiple instances of the PEX Inductance ... Frequency statement for different nets. For skin effect modeling, you can set the net frequency up to 50 GHz.

### Note



If the PEX Inductance ... Frequency statement and the [PEX Inductance Switch Time](#) statements are specified for the same net in the rule file, the Calibre xL tool uses broadband model and the highest frequency specified in the statements.

## Examples

### Example 1

The following statement instructs the Calibre xL tool to use an operating frequency of 10 GHz for net\_1.

```
PEX INDUCTANCE SOURCENAMES net_1 FREQUENCY 1.0e10
```

### Example 2

The following statement instructs the Calibre xL tool to use a maximum operating frequency of 10 GHz for net\_1.

```
PEX INDUCTANCE SOURCENAMES net_1 MAXIMUM FREQUENCY 1.0e10
```

# PEX Inductance Layer Summary

Parasitic extraction

## PEX INDUCTANCE LAYER SUMMARY {ON | OFF}

Used only in Calibre xL.

### Parameters

- **ON**

A required keyword which instructs Calibre xL to output a layer summary showing the layers used, their physical features and LVS name to the transcript.

- **OFF**

A required keyword that suppresses the report from the transcript. This is the default behavior.

### Description

This statement allows the user to control the output of the layer summary data to the transcript. The default setting OFF as this information may only be useful for debugging. You can specify this statement once in your SVRF rule file.

### Examples

To output the layer summary to the transcript, use the following:

```
PEX INDUCTANCE LAYER SUMMARY ON
```

The resulting layer summary will look like:

```
-----
LAYER SUMMARY:
Name      Height      Thickness      Conductivity
M1       1.6100e+03 6.0000e+02  2.3810e+00
M2       2.9460e+03 1.0000e+03  2.0000e+00
```

## PEX Inductance MICHECK Constraint

Parasitic extraction

### PEX INDUCTANCE MICHECK CONSTRAINT {ON | OFF}

Used only in Calibre xL.

#### Parameters

- **ON**  
A required keyword which instructs Inductance ERC to only output mutual couplings that validate the MICHECK rules. This is the default behavior.
- **OFF**  
A required keyword that causes Inductance ERC to ignore the constraints specified in the inductance MICHECK rules and report all couplings.

#### Description

This statement allows the user to control the output of the mutual couplings. The default setting ON will only output mutual couplings that validate the MICHECK rules. Turning off the constraint ignores the constraints specified in the Inductance MICHECK rules and reports all couplings. You can specify this statement once in your SVRF rule file.

#### Examples

To ignore the constraints specified in the Inductance MICHECK rules and report all couplings, use the following:

```
PEX INDUCTANCE MICHECK CONSTRAINT OFF
```

# PEX Inductance Minlength

Parasitic extraction

## PEX INDUCTANCE MINLENGTH *value*

Used only in Calibre xL.

### Parameters

- *value*

A required parameter that specifies the minimum path length in microns. Any signal path with length larger than *value* will have inductance computed. Paths with length less than or equal to *value* will use the default per unit length inductance value given in [PEX Inductance Self](#). The default is 100 if you do not specify this statement.

### Description

Specifies the minimum path length used to accurately compute self inductance.

### Examples

#### Example 1

The following statement explicitly instructs the Calibre xL tool to accurately compute inductance for all paths, no matter what their lengths are.

```
PEX INDUCTANCE MINLENGTH 0
```

#### Example 2

The following statement instructs the Calibre xL tool to accurately compute inductance for path lengths greater than 20 microns.

```
PEX INDUCTANCE MINLENGTH 20
```

## PEX Inductance Parameters

Parasitic extraction

**PEX INDUCTANCE PARAMETERS** *layer {BASE inductance\_base}*  
  *{RADIUS inductance\_radius}*

Used only in Calibre xL.

### Parameters

- *layer*  
A required argument specifying the via layer.
- **BASE inductance\_base**  
A required keyword and value, in inductance units, specifying the inductance of a via on the via layer with unit radius, where radius is equal to 1 micron. The *inductance\_base* value is used as the B value in the equation  $L = B - R * \log_e(r)$ .
- **RADIUS inductance\_radius**  
A required keyword and value, in inductance units, specifying the inductance per  $\log_e(r)$ , where  $r$  is the square root of the area (the overlapping region between the top and bottom layers of the via) divided by two. The *inductance\_radius* value is used as the R value in the equation  $L = B - R * \log_e(r)$ .

### Description

This optional statement specifies the parameters to use in the following equation for calculating the self inductance of vias:

$$L = B - R * \log_e(r)$$

where L is self inductance, B is *inductance\_base*, R is *inductance\_radius*, and r is the square root of the area (the overlapping region between the top and bottom layers of the via) divided by two.  $\log_e$  is the mathematical symbol representing natural logarithm functions. Values are specified in inductance units, which defaults to picohenries and is specified using the [Unit Inductance](#) statement.

### Examples

The following statement instructs the Calibre xL tool to calculate the self inductance of vias on the layer via based on the parameter value 1.0 for B and 0.02 for R in the equation  $L = B - R * \log_e(r)$ .

```
PEX INDUCTANCE PARAMETERS VIA BASE 1.0 RADIUS 0.02
```

# PEX Inductance Range

Parasitic extraction

## PEX INDUCTANCE RANGE *distance*

Used only in Calibre xL.

### Parameters

- *distance*

A required parameter that specifies the maximum distance in microns to search for the return paths. The format is a floating-point number expressed in scientific notation. Specifying a distance of 0 will cause Calibre xL to revert to the default, which means it will search the extent of the chip. This behavior is the same as what occurs when the statement is not in the rule file.

### Description

Specifies how far to search from a given net segment for the return paths to include in self impedance calculations.

### Examples

#### Example 1

The following statement explicitly instructs the Calibre xL tool to search the extent of the chip for return paths to include in self impedance calculations.

```
PEX INDUCTANCE RANGE 0
```

#### Example 2

The following statement instructs the Calibre xL tool to restrict the search to 20 microns for return paths to include in self impedance calculations.

```
PEX INDUCTANCE RANGE 20
```

## PEX Inductance Returnpath

Parasitic extraction

**PEX INDUCTANCE RETURNPATH {GROUND | **POWER** | **GANDP** | **GANDPAVG** | **NONE**}**

Used only in Calibre xL.

### Parameters

- **GROUND**

A required keyword that considers only ground nets as the possible return path. This is the default if the statement is not specified.

- ****POWER****

A required keyword that considers only power nets as the possible return path.

- ****GANDP****

A required keyword that considers only power and ground nets as the possible return path.

- ****GANDPAVG****

A required keyword that computes two different values (one with GROUND as the return path and the other with POWER as the returnpath) and reports the average of the two as the actual impedance value.

- ****NONE****

---

**Note**



As of the 2010.3 release, NONE keyword has been deprecated and cannot be used for differential pair extraction. For more information on differential pair extraction using the standard mutual inductance flow, see “[Differential Pair Extraction](#)” in the *Calibre xL User’s Manual*.

---

A required keyword that specifies to only compute differential pairs.

### Description

Specifies the type of nets (ground or power) that may be considered as candidates for return path in inductance extraction. You can use the [PEX Ground](#) and [PEX Power](#) statements to specify the ground and power nets.

---

**Note**



During a compound run — that is, a run in which inductance is extracted in conjunction with resistance and capacitance — return paths are not excluded regardless of whether or not they are specified as excluded nets (for example, in the [PEX Extract Exclude](#) statement). If you are using the -select switch, the Calibre xL tool will extract return path nets if they have not already been extracted.

---

## Examples

The following statement instructs the Calibre xL tool to only consider power nets as candidates for return path when extracting self loop impedance and loop resistance of a signal.

```
PEX INDUCTANCE RETURNPATH POWER
```

## PEX Inductance Same Net Mutual

Parasitic extraction

### PEX INDUCTANCE SAME NET MUTUAL {ON | OFF}

Used only in Calibre xL.

#### Parameters

- **ON**  
A required keyword that enables mutual inductance extraction between wire segments that are part of the same net.
- **OFF**  
A required keyword that disables mutual inductance extraction between wire segments that are part of the same net. This is the default if the statement is not specified.

#### Description

Enables the extraction of the mutual inductance between wire segments within the same net, such as in stacked metal configuration. This additional computation is performed during the self inductance extraction stage.

#### Examples

Include the following statement in your rule file to extract the mutual inductance between wire segments within the same net.

```
PEX INDUCTANCE SAME NET MUTUAL ON
```

# PEX Inductance Self

Parasitic extraction

## PEX INDUCTANCE SELF *value*

Used only in Calibre xL.

### Parameters

- *value*

A required parameter that specifies the self inductance value in nanohenries per micron. The default is 0 if you do not specify this statement.

### Description

Specifies the self inductance value to use on filtered paths.

### Examples

The following statement instructs the Calibre xL tool to use 2 picohenries per micron for the self inductance of filtered paths.

```
PEX INDUCTANCE SELF 2e-3
```

## PEX Inductance Skin Include

Parasitic extraction

**PEX INDUCTANCE SKIN INCLUDE** [LAYOUTNAMES | SOURCENAMES] *net\_name*  
[*net\_name* ...]

Used only in Calibre xL.

### Parameters

- LAYOUTNAMES

An optional keyword that specifies the net names are derived from the layout. This is the default if the statement is not specified.

- SOURCENAMES

An optional keyword that specifies the net names are derived from the schematic. Ensure the SVRF rule file contains a valid Source Path statement and a valid Source Primary statement identifying the primary cell when specifying this option.

- ***net\_name***

A required name identifying the net. You can add more than one net name.

### Description

Specifies one or more nets for which you want skin effect calculated.

### Examples

The following example instructs the Calibre xL tool to calculate skin effect for net1 and net3.

```
PEX INDUCTANCE SKIN INCLUDE net1 net3
```

## PEX Inductance Switch Time

Parasitic extraction

### PEX INDUCTANCE SWITCH TIME *seconds*

Used only in Calibre xL.

#### Parameters

- *seconds*

A required value that specifies the rise or fall time of all nets. The format is a floating-point number in scientific notation. Use the shorter of the rise or fall time.

#### Description

Specifies the global switch time to use during inductance extraction for all nets in the design. It instructs the Calibre xL tool to generate a broadband netlist. For skin effect modeling, you can set the frequency up to 50 GHz (approximately 6.5 picoseconds). A broadband netlist is only generated for paths that are at least 100 microns long. You can override this statement for specific nets for which the switch time is different from the global switch time using the [PEX Inductance ... Switch Time](#) statement.

---

#### Note



If both the PEX Inductance Switch\_Time and the [PEX Inductance Frequency](#) statements are specified in the rule file, the Calibre xL tool uses the broadband model and the highest frequency specified in the statements.

---

#### Examples

The following statement instructs the Calibre xL tool to use a switch time of 100 ps for all nets in the design during inductance extraction.

```
PEX INDUCTANCE SWITCH TIME 1.0e-10
```

## PEX Inductance Switch\_Time

Parasitic extraction

**PEX INDUCTANCE SWITCH\_TIME** *seconds*

---

**Note**

 PEX Inductance Switch\_Time has been renamed to [PEX Inductance Switch Time](#). There are no changes to parameter definitions and functionality. PEX Inductance Switch\_Time has been deprecated as of the 2009.1 release.

---

# PEX Inductance ... Switch Time

Parasitic extraction

## Syntax

**PEX INDUCTANCE [LAYOUTNAMES | SOURCENAMES] *net\_name* SWITCH TIME  
*seconds***

## Parameters

- **LAYOUTNAMES**  
 An optional keyword that specifies the net names are derived from the layout.
- **SOURCENAMES**  
 An optional keyword that specifies the net names are derived from the schematic. When specifying SOURCENAMES, ensure the SVRF rule file contains a valid Source Path statement and a valid Source Primary statement identifying the primary cell.
- ***net\_name***  
 A required argument specifying the name of the net to be used for analysis.
- ***seconds***  
 A required argument that specifies the rise or fall time of the net. The format is a floating-point number in scientific notation. Use the shorter of the rise or fall time.

## Description

Specifies the nets for which the switch time is different from the global switch time specified with the [PEX Inductance Switch Time](#) statement. It instructs the Calibre xL tool to generate a broadband netlist for the specified net. You can specify multiple instances of this statement for different nets. For skin effect modeling, you can set the frequency up to 50 GHz (approximately 6.5 picoseconds). A broadband netlist is only generated for paths that are at least 100 microns long.

## Examples

### Example 1

The following statement instructs the Calibre xL tool to use a switch\_time of 100 picoseconds for net\_1.

```
PEX INDUCTANCE SOURCENAMES net_1 SWITCH TIME 1.0e-10
```

### Example 2

The following statement instructs the Calibre xL tool to generate a broadband netlist for capturing broadband behavior of impedance for net\_1. To model effects at an intermediate

## **PEX Inductance ... Switch Time**

---

frequency, a broadband netlist is generated using self impedance calculated at a low frequency (less than 100 MHz) and at a high frequency specified by:

$$\frac{1}{\pi(10^{-10})} \approx 3.2GHz$$

```
PEX INDUCTANCE SOURCENAMES net_1 SWITCH TIME 1.0e-10
```

## PEX Inductance ... Switch\_Time

Parasitic extraction

**PEX INDUCTANCE** [LAYOUT | SOURCE] *net\_name* **SWITCH\_TIME** *seconds*

---

**Note**

 PEX Inductance ... Switch\_Time has been renamed to [PEX Inductance ... Switch Time](#).

There are no changes to parameter definitions and functionality. PEX Inductance ... Switch\_Time has been deprecated as of the 2009.1 release.

---

## PEX Inductance Victim

Parasitic extraction

**PEX INDUCTANCE VICTIM [LAYOUTNAMES | SOURCENAMES] *net\_name***

[*pin\_name pin\_tag*] [*tube\_radius*]

Used only in Calibre xL.

### Parameters

- **LAYOUTNAMES**

An optional keyword that specifies the net names are derived from the layout. This is the default.

- **SOURCENAMES**

An optional keyword that specifies the net names are derived from the schematic. The SVRF rule file must contain a valid **Source Path** statement and a valid **Source Primary** statement identifying the primary cell when specifying this option.

- ***net\_name***

A required parameter that specifies the name of the victim net.

- ***pin\_name pin\_tag***

An optional parameter pair that specifies the driver on the net. These parameters should match the pin list or port list as specified by the Calibre RVE viewer. All tags must be in lower case. These parameters provide the driver on the net so the Calibre xL tool can perform a path-wise analysis of the net.

- ***tube\_radius***

An optional value that specifies the radius of the tube around the victim path. The default unit is microns.

### Description

Identifies a victim net and provides information that identifies the victim paths and the aggressors on the victim paths. The Calibre xL tool uses this information to analyze the victim net and calculate mutual impedance to aggressors.

Include this statement in a user defined ASCII file named *victim.xl*. Add a statement to this file for each victim net that should be considered for mutual impedance extraction. Specify the location of the *victim.xl* file using the **PEX Inductance Victim File** statement. PEX Inductance Victim statements can also be directly specified in the SVRF rule file.

---

**Note**



If you are using a driver and receiver file, the driver specified in the file is used instead of the driver specified in the PEX Inductance Victim statement.

---

## Examples

The file myVictims.xl will contain the following statement that instructs the Calibre xL tool to analyze net IP as a victim and calculate mutual impedance to aggressors.

```
PEX INDUCTANCE VICTIM IP X117/X4/X0/X0/X0/X1/M0 d 60
```

## PEX Inductance Victim Path

Parasitic extraction

**PEX INDUCTANCE VICTIM PATH** [LAYOUTNAMES | SOURCENAMES] *net\_name*  
    *driver\_pin\_name driver\_pin\_tag receiver\_pin\_name receiver\_pin\_tag* [*tube\_radius*]

Used only in Calibre xL.

### Parameters

- LAYOUTNAMES

An optional keyword that specifies the net names are derived from the layout. This is the default.

- SOURCENAMES

An optional keyword that specifies the net names are derived from the schematic. The SVRF rule file must contain a valid [Source Path](#) statement and a valid [Source Primary](#) statement identifying the primary cell when specifying this option.

- ***net\_name***

A required parameter that specifies the name of the net containing the victim path.

- ***driver\_pin\_name driver\_pin\_tag***

A required parameter and value pair that specifies the layout pin or port pair used to identify where the victim path begins, and the driver tag for the pin or port.

- ***receiver\_pin\_name receiver\_pin\_tag***

A required parameter and value pair that specifies the layout pin or port pair used to identify where the victim path ends, and the receiver tag for the pin or port.

- ***tube\_radius***

An optional value that specifies the radius of the tube around the victim path. This parameter overrides the value specified using the PEX\_MUTIN\_RANGE environment variable. The default unit is microns.

### Description

Specifies the victim path associated with a victim net. This statement may be included in the victim file along with the victim net names, or in the SVRF rule file.

---

**Note**



If you are using a driver and receiver file, the driver specified in the file is used instead of the driver specified in the PEX Inductance Victim Path statement.

---

## Examples

The following statement instructs the Calibre xL tool to search for the victim path associated with the net si8. The path begins at the pin/port named si8 with a tag “t” (text port) and ends at the pin/port named so8 with the tag “t” (text port).

```
PEX INDUCTANCE VICTIM PATH si8 si8 t so8 t
```

## PEX Inductance Victim\_Path

Parasitic extraction

**PEX INDUCTANCE VICTIM\_PATH** *id net\_name driver\_pin\_name driver\_pin\_tag  
receiver\_pin\_name receiver\_pin\_tag*

---

**Note**



PEX Inductance Victim\_Path has been renamed to [PEX Inductance Victim Path](#). There are no changes to parameter definitions and functionality. PEX Inductance Victim\_Path has been deprecated as of the 2009.2 release.

---

# PEX Inductance Victim File

Parasitic extraction

## PEX INDUCTANCE VICTIM FILE *victim\_file\_location*

Used only in Calibre xL.

### Parameters

- *victim\_file\_location*

A required parameter that specifies the location of the victim file. You can use an empty string { " " } if needed. For example, if you have a victim file in the working directory and want to run differential pairs, use an empty string for the *victim\_file\_location*.

### Description

Specifies the location of the victim file. For more information on creating a victim file, see the the [PEX Inductance Victim](#) statement.

### Examples

The following statement instructs the Calibre xL tool to search for the victim file in the working directory.

```
PEX INDUCTANCE VICTIM FILE "./victim.xl"
```

## **PEX Inductance Victim\_File**

Parasitic extraction

**PEX INDUCTANCE VICTIM\_FILE** *victim\_file\_location*

---

**Note**

 PEX Inductance Victim\_File has been renamed to [PEX Inductance Victim File](#). There are no changes to parameter definitions and functionality. PEX Inductance Victim\_File has been deprecated as of the 2009.1 release.

---

# PEX Magnify

Parasitic extraction

**PEX MAGNIFY** *shrink\_value* [PROPERTY]

## Parameters

- *shrink\_value*

A required, positive, floating-point number that specifies the amount by which to scale the layout.

- PROPERTY

An optional keyword that indicates to scale the device parameters.

## Description

This statement specifies to scale the layout before parasitic extraction. This occurs before in-die variation mapping. The *shrink\_value* parameter is applied to a polygon's width, length, and spacing properties.

## Examples

You have a 130 nm design that you want to use with a 110 nm process. In the 130 nm process, the minimum width of metal1 is 0.130 microns. In the 110 nm process, the minimum width of metal1 is 0.110 microns (eighty-five percent of 130 nm), so you need to use a shrink value of 0.85 to shrink your 130 nm design 85 percent.

```
PEX MAGNIFY 0.85
```

With a shrink value of 85 percent, the 0.130 micron line is scaled to the drawn value of 0.110 microns ( $0.130 \times 0.85 = 0.110$ ).

## PEX Map

Parasitic extraction

**PEX MAP** *calibrated\_layer\_name physical\_layer\_name [physical\_layer\_name...]*

### Parameters

- ***calibrated\_layer\_name***

A required parameter that specifies the name of the calibrated layer to be mapped.

- ***physical\_layer\_name***

A required parameter that specifies one or more original layer names or derived layer names that will inherit rules from the calibrated layer. These layers must be of the proper type and have appropriate connectivity defined for the rules they are inheriting.

### Description

This statement defines the mapping between a calibrated layer and one or more physical layers. Rules for the calibrated layer will be applied to each physical layer.

A calibrated layer may be mapped only once. Any physical layer can be mapped to only one calibrated layer. If a calibrated layer is not explicitly mapped, then there is an implicit mapping by name.

All physical layers listed in a PEX Map statement must be co-planar.

Do not use this statement with [PEX Alias](#). Any layer specified in both a PEX Map and PEX Alias statement will generate an error and extraction will stop.

### Examples

#### Example 1

To map calibrated layer METAL1 to physical layers METAL1 and MET1a, include the following statement in the rule file:

```
PEX MAP METAL1 METAL1 MET1a
```

#### Example 2

To map calibrated layer METAL1 to the physical layer MET1a, include the following statement in the rule file:

```
PEX MAP METAL1 MET1a
```

If there is a physical layer METAL1, it will not inherit rules from the calibrated layer METAL1 even though they have the same name.

#### Example 3

If a calibrated layer is not explicitly mapped to any physical layer with the PEX Map statement, then there is an implied mapping by name. The following example defines the default behavior for METAL1:

```
PEX MAP METAL1 METAL1
```

Calibrated layer METAL1 would be implicitly mapped to physical layer METAL1 without including this statement in the rule file.

#### Example 4

If a calibrated layer is mapped to a physical layer that does not exist, then the associated rules for the calibrated layer will not be applied to any physical layer:

```
PEX MAP METAL1 FOO
```

In this case physical layer FOO does not exist and calibrated layer METAL1 will not be mapped to any physical layer.

## PEX Netlist

Parasitic extraction

### Summary

Generates a netlist with parasitic elements and places it in the file specified. Used for all extracted netlist creation except ADMS format ([PEX Netlist ADMS](#)) and non-parasitic format ([PEX Netlist Simple](#)).

Syntax 1, generating HSPICE netlists:

```
PEX NETLIST filename HSPICE [scale] [ADITRAIL] {LAYOUTNAMES |  
  SOURCENAMES | SOURCEBASED | SCHEMATICONLY [filename]}  
  [COORDSCALE value] [GROUND ground_name] [LISTPROBES] [LOCATION]  
  [NOINSTANCEX] [PINNAMESEP character] [PRUNE] [RACTUAL] [RAREA]  
  [RCNAMED] [RLAYER] [RLENGTH] [RLOCATION] [RSCALE value]  
  [RTHICKNESS] [RVIACOUNT] [RWIDTH] [SEPARATOR sep_name]  
  [SHORTPINNAMES] [SINGLEFILE] [TCOEFF]
```

Syntax 2, generating DSPF netlists:

```
PEX NETLIST filename DSPF [scale] [PRIMETIME | HSIM | ADITRAIL]  
  {LAYOUTNAMES | SOURCENAMES | SOURCEBASED | SCHEMATICONLY  
  [filename]} [COORDSCALE value] [GROUND ground_name] [LOCATION]  
  [NOINSTANCESECTION] [NOINSTANCEX] [NOTOPSUBCKT] [PRUNE]  
  [RACTUAL] [RAREA] [RCNAMED] [RLAYER] [RLENGTH] [RLOCATION]  
  [RSCALE value] [RTHICKNESS] [RWIDTH] [SEPARATOR sep_name] [TCOEFF]
```

Syntax 3, generating SPEF netlists:

```
PEX NETLIST filename SPEF [PRIMETIME] {LAYOUTNAMES | SOURCENAMES |  
  SOURCEBASED | SCHEMATICONLY [filename]} [BUSDELIM string]  
  [COORDSCALE value] [GROUND ground_name] [LOCATION] [MAPNAMES]  
  [NOINSTANCEX] [RACTUAL] [RAREA] [RLAYER] [RLENGTH] [RLOCATION]  
  [RSCALE value] [RTHICKNESS] [RWIDTH] [SEPARATOR sep_name]
```

Syntax 4, generating SPECTRE netlists:

```
PEX NETLIST filename SPECTRE [scale] {LAYOUTNAMES | SOURCENAMES |  
  SOURCEBASED | SCHEMATICONLY [filename]} [COORDSCALE value]  
  [GROUND ground_name] [LISTPROBES] [LOCATION] [NOINSTANCEX]  
  [RACTUAL] [RLAYER] [RLENGTH] [RLOCATION] [RSCALE value] [RTHICKNESS]  
  [RWIDTH] [SEPARATOR sep_name] [SINGLEFILE] [TCOEFF]
```

Syntax 5, generating ELDO netlists:

```
PEX NETLIST filename ELDO [scale] {LAYOUTNAMES | SOURCENAMES |  
  SOURCEBASED | SCHEMATICONLY [filename]} [COORDSCALE value]  
  [GROUND ground_name] [LISTPROBES] [LOCATION] [PRUNE] [RACTUAL]  
  [RCNAMED] [RLAYER] [RLENGTH] [RLOCATION] [RSCALE value]  
  [RTHICKNESS] [RWIDTH] [SEPARATOR sep_name] [SINGLEFILE] [TCOEFF]
```

Syntax 6, generating CALIBREVIEW output:

```
PEX NETLIST filename CALIBREVIEW [scale] [ADITRAIL] {LAYOUTNAMES |  

SOURCENAMES | SOURCEBASED | SCHEMATICONLY [filename]}  

[COORDSCALE value] [ELDO_KR] [GROUND ground_name] [ICELLMAP]  

[LOCATION] [RACTUAL] [RLAYER] [RLENGTH] [RLOCATION] [RSCALE value]  

[RTHICKNESS] [RWIDTH] [SEPARATOR sep_name]
```

## Parameters

- *filename*

A required filename for the created netlist.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “Key Concepts”.

If the *filename* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. For HSPICE and ELDO formats, each file is individually compressed. The references in the main netlist to the other files includes the suffix. This functionality assumes the compress and gzip commands are available in your \$PATH.

- One of the following must be specified. Possible choices are:

**HSPICE** [*scale*] [**ADITRAIL**] — The netlist format is HSPICE.

**DSPF** [*scale*] [**PRIMETIME** | **HSIM** | **ADITRAIL**] — The netlist format is DSPF.

**SPEF** [**PRIMETIME**] — The netlist format is SPEF.

**SPECTRE** [*scale*] — The netlist format is Spectre.

**ELDO** [*scale*] — The netlist format is Eldo.

**CALIBREVIEW** [*scale*] [**ADITRAIL**] — The netlist format is CalibreView.

The optional string *scale* must be a positive floating-point number and may be a variable. It specifies the size multiplier (in units of meters) for the L, W, A, P, AS, AD, PS, and PD parameters of MOSFET, diode, and JFET devices. Set *scale* to 1e-6 meters to enter the parameters in microns; areas are then in square microns. The default value is 1.

The optional keyword PRIMETIME creates netlist that PrimeTime® and other static timing analysis tools will be able to parse. This keyword is supported in nearly all modes of extraction. Distributed RCC mode extraction (-rcc) is not supported for DSPF in PrimeTime. Distributed RC mode is supported.

When PRIMETIME is specified, Xs are removed in front of device names and keywords and element names are in all uppercase. Additionally, in DSPF, the instance names are shortened.

The optional keyword HSIM specifies settings for producing a netlist suitable for use with Synopsys HSIMplus Co-Simulation. When HSIM is specified, feedthrough nets are

modeled with in the cell and any extracted coupling caps are listed by NET and removed from the Instance section. This parameter setting is only used with the DSPF output format.

The optional keyword ADITRAIL specifies settings for producing a netlist suitable for use with Mentor Graphics fast-SPICE simulation tool ADiT Rail. Specifying ADITRAIL includes layer data, parasitic data, port, device and parasitic location information in the netlist. The resulting netlist will contain additional resistors with default value of 1.01e-05. These resistors are used to establish connections between parts or the resistor network, maintaining un-distorted layout coordinates of the resistors. This parameter setting is only used with the CalibreView, DSPF, and HSPICE output formats.

- One of the following must be specified. Possible choices are:

**LAYOUTNAMES** — Specifies the names are derived from the layout.

**SOURCENAMES** — Specifies the names are derived from the schematic or netlist, but the hierarchy is based on the layout. When specifying the **SOURCENAMES** option, make sure that a valid [Source Primary](#) statement identifying a valid primary cell is in the rule file. Also be sure that the naming conventions used in the source are compatible with your output netlist format.

**SOURCEBASED** — Specifies the formatter to use the circuit pin order and cell hierarchy in the source netlist instead of the layout netlist when generating the extracted netlist. If you are creating the source netlist with the Calibre nmLVS software, you must use nmLVS-H. The design must pass LVS.

**SCHEMATICONLY** [*filename*] — Specifies a parasitic netlist based on the LVS source hierarchy, with intentional device parameters extracted and backannotated from the layout. Parasitics are netlisted only for nets that are cross-referenced to the source netlist. If *filename* is not used, only standard parameters for MOSFETs are backannotated. Other devices use source parameters. The *filename* parameter specifies a file containing entries in the following format for defining device parameters:

```
ELEMENT element_name MODEL model_name
[
    prop_type1 = val1
    prop_type2 = val2
    ...
    prop_typeN = valN
]
```

where:

- ELEMENT is a required keyword, correlated to the DEVICE statements found in the LVS rules.
- *element\_name* is the LVS standard device name, such as R for resistor, C for capacitor, and so forth. [Table 4-5](#) shows a complete list of available default devices.

- MODEL is an optional keyword. If this keyword is missing from any element, then the same calculation will be applied to all elements without MODEL specified.
- *model\_name* is the device model name, such as RP for poly resistor, RN for n-well resistor, and so forth.
- *prop\_type* is the parameter that will be calculated for the device.
- *val* can be set to:

min	use minimum value from layout
max	use maximum value from layout
median	use median value from layout
sum	add layout values for parameter and divide by source device count
ave	add layout values for parameter and divide by layout device count
source	use value from source

The input file may contain comments. Comments begin with // and run to the end of the line. The square brackets are not optional and must be included.

- **BUSDELIM *string***

Keyword used with SPEF output format that specifies an optional string to set the BUS\_DELIMITER variable in the SPEF header. The *string* value must be enclosed in quotes (""). The default is a set of square brackets ("[ ]").

- **COORDSCALE *value***

An optional keyword set, where *value* is a scaling factor specified in meters. The X,Y coordinates in the netlist are scaled to the units specified by *value*. If *value* is 1.0, then coordinates are reported in meters. If *value* is 1.0e-06, then coordinates are reported in microns. The default setting for *value* is in units (dbu/precision), which is typically microns.

- **ELDO\_KR**

An optional keyword used to add the R parameter to L instances, and the KR parameter to K instances in the CalibreView netlist. The formatter does not add any f or v elements for mutual resistance. Use only with the CalibreView netlist format.

- **GROUND *ground\_name***

An optional keyword set, where *ground\_name* is the name of the ground node. All capacitances are grounded to this node. The default value of *ground\_name* is 0. If the *ground\_name* is not a global net, you may use [PEX Netlist Global Nets](#) to make it global.

- ICELLMAP

An optional keyword used to output the subtype name, if the device subtype is available. If device subtype is not available, then output the built-in model name. Use only with the CalibreView netlist format.

- LISTPROBES

An optional keyword used to include probes in the .SUBCKT definitions for HSPICE, Eldo, and Spectre formats.

- LOCATION

An optional keyword that specifies to report device locations as comments. This includes the locations for intentional resistors and capacitors. LOCATION keyword does not work with the **SCHEMATICONLY** and **SOURCEBASED** parameters.

- MAPNAMES

Keyword used with SPEF output format that specifies to generate a name map. The name map is a list that maps instance names to numerical identifiers. The identifiers are printed in place of the instance names in the netlist's model sections; this produces more compact SPEF netlists.

- NOINSTANCESECTION

Keyword used with DSPF output format that specifies eliminating the instance section from the DSPF output. If NOINSTANCESECTION is not included in the PEX Netlist declaration, the instance section is included. Excluding the instance section can save execution time for a case such as backannotation to Verilog.

- NOINSTANCEX

Keyword that causes the formatter to filter Xs in HSPICE, DSPF, SPEF, and Spectre output. Turn on filtering by including NOINSTANCEX in the PEX Netlist statement in your rule file.

- NOTOPSUBCKT

Keyword used with DSPF output format that suppresses the top-level subcircuit in the output netlist from the formatter. This is useful if your simulation test bench already has the top-level subcircuit and you want to include the parasitics, or if you have trouble producing netlists that can be simulated for PrimeTime.

- PINNAMESEP *character*

Keyword that defines the character which the formatter places between device pins and parasitic model subcircuits when creating net names. The character must occur between quotation marks (""). The default is “\_”.

- PRUNE

An optional keyword that removes all intentional devices not attached to the extracted nets. Use this keyword with selected net extraction as described in “[Netlisting Only Direct Devices on a Selected Net](#)” in the *Calibre xRC User’s Guide*. Used only with the HSPICE, DSPF, and ELDO netlist formats.

- **RACTUAL**

An optional keyword that specifies to report actual parasitic resistor width dimensions in the netlist when used with RWIDTH. If RACTUAL is not specified then the drawn width values are reported when using RWIDTH. Use with calibrated rule files generated by 2010.1 and later releases of xCalibrate.

- **RAREA**

An optional keyword that specifies to report contact/via area used for extracting parasitic resistors. The value is reported as a comment. RAREA is only supported with the DSPF, HSPICE, and SPEF formats.

- **RCNAMED**

An optional keyword that causes the R or C value of an intentional device to be replaced with a parameter (“R=*value*” or “C=*value*”) in HSPICE, DSPF, or Eldo netlists for R or C devices that have models.

- **RLAYER**

An optional keyword that specifies to report parasitic resistor layers as comments.

Do not use RLAYER when performing resistance reduction. Using RLAYER with [PEX Reduce ROnly](#), [PEX Reduce TICER](#), [PEX Reduce Digital](#), [PEX Reduce Analog](#), or [PEX Reduce Minres](#) COMBINE will generate a warning and RLAYER will be ignored.

- **RLENGTH**

An optional keyword that specifies to report parasitic resistor lengths as comments.

Do not use RLENGTH when performing resistance reduction. Using RLENGTH with [PEX Reduce ROnly](#), [PEX Reduce TICER](#), [PEX Reduce Digital](#), [PEX Reduce Analog](#), or [PEX Reduce Minres](#) COMBINE will generate a warning and RLENGTH will be ignored.

- **RLOCATION**

An optional keyword that specifies to report parasitic resistor locations as comments. For CalibreView format netlists, it also reports parasitic capacitor locations.

Do not use RLOCATION when performing resistance reduction. Using RLOCATION with [PEX Reduce ROnly](#), [PEX Reduce TICER](#), [PEX Reduce Digital](#), [PEX Reduce Analog](#), or [PEX Reduce Minres](#) COMBINE will generate a warning and RLOCATION will be ignored.

- **RSCALE *value***

An optional keyword set, where *value* is a scaling factor that changes parasitic resistor properties, such as width, independently of device properties. The default value is 1.

- **RTHICKNESS**

An optional keyword that specifies to report calculated parasitic resistor thicknesses as comments. When resistance is specified with [Resistance Sheet](#), thickness is not reported.

Do not use RTHICKNESS when performing resistance reduction. Using RTHICKNESS with [PEX Reduce ROnly](#), [PEX Reduce TICER](#), [PEX Reduce Digital](#), [PEX Reduce Analog](#),

or [PEX Reduce Minres COMBINE](#) will generate a warning and RTHICKNESS will be ignored.

- **RVIACOUNT**

An optional keyword that causes the formatter to report the number of vias/contacts represented by the netlisted via resistor value. The value is reported as a comment. Use only with the HSPICE netlist format.

- **RWIDTH**

An optional keyword that specifies to report parasitic resistor widths as comments.

Do not use RWIDTH when performing resistance reduction. Using RWIDTH with [PEX Reduce ROnly](#), [PEX Reduce TICER](#), [PEX Reduce Digital](#), [PEX Reduce Analog](#), or [PEX Reduce Minres COMBINE](#) will generate a warning and RWIDTH will be ignored.

- **SEPARATOR *sep\_name***

An optional keyword set, where *sep\_name* allows you to replace the default hierarchy separator slash (/) with *sep\_name* in any net or instance name from the source or layout. If the first character in the name is a slash, it is deleted. Note that for Eldo netlist format the default hierarchy separator is underscore (\_).

- **SHORTPINNAMES**

Keyword used with HSPICE output format that causes the formatter to replace pin names with an integer when creating SPICE net names.

- **SINGLEFILE**

An optional keyword that specifies to combine the transistor-level SPICE data, the subcircuit definitions of the parasitic net models for each extracted net, the parasitic resistors, the parasitic capacitors, and the instances of the net models into a single parasitic netlist output file. Used only with the HSPICE, ELDO, and Spectre netlist formats.

- **TCOEFF**

An optional keyword that specifies to output temperature coefficients for parasitic resistors for the DSPF, Eldo, HSPICE, and Spectre formats. It is ignored for CalibreView and SPEF.

## Description

Generates a netlist for the extracted circuit with distributed parasitic elements and placed in the file specified. The **LAYOUTNAMES**, **SOURCENAMES**, **SOURCEBASED**, and **SCHEMATICONLY** parameters can be used with all output formats. Table 4-41 shows how the device names from layout and source are handled for each of these options.

**Table 4-41. Device Name, Net Name, and Parameter Conventions in Netlist**

Option	Device/Net Name	Device Parameter
LAYOUTNAMES	Layout devices and nets with layout names	Layout
SOURCENAMES	Layout devices and nets with source names	Layout
SOURCEBASED	Source devices and nets with source names	Source

**Table 4-41. Device Name, Net Name, and Parameter Conventions in Netlist**

Option	Device/Net Name	Device Parameter
SCHEMATICONLY	Source devices and nets with source names	Layout

### Using Parasitic Resistance Parameters

Specify the RWIDTH, RLENGTH, RVIACOUNT, RAREA, RLAYER, and RLOCATION options if you want Calibre xRC to include width, length, via/contact count, via/contact area, layer data, and location information in your parasitic databases. These cause SVDBs and parasitic netlists to be larger.

#### Note



If reduction modifies parasitic resistors, the Calibre xRC tool may generate invalid area, location, width, length, and layer information.

The output netlist results are:

- RWIDTH: \$w=*number*
- RLENGTH: \$l=*number*
- RVIACOUNT: \$count=*number*
- RAREA: \$a=*number*
- RLAYER: \$layer=*layer\_name*
- RLOCATION: \$x=*x\_coord* \$y=*y\_coord* \$x2=*x2\_coord* \$y2=*y2\_coord*

The x and y coordinates correspond to the x and y coordinates of the nodes to which the parasitic resistor is connected. The parameters must be present in the rule file before extracting parasitics at the time you invoke calibre -xrc -pdb to store the count, area, location, width, length, and layer information in the SVDB.

Several keyword combinations are possible for displaying physical properties of parasitic devices. [Table 4-42](#) shows the possible combinations and the resulting outputs.

**Table 4-42. PEX Netlist Keyword Combinations and Results**

Keywords	Displayed Results
RAREA	Only area of contact and via resistors is displayed.
RAREA, RWIDTH	Area of contact and via resistors and width of other extracted resistors.
RAREA, RLENGTH	Area of contact and via resistors and length of other extracted resistors.
RAREA, RLAYER	Area of contact and via resistors and layers for all extracted resistors.
RAREA, RWIDTH, RLAYER	Area of contact and via resistors, width for other extracted resistors, layers for all extracted resistors.
RAREA, RLENGTH, RLAYER	Area of contact and via resistors, length for other extracted resistors, layers for all extracted resistors.

**Table 4-42. PEX Netlist Keyword Combinations and Results**

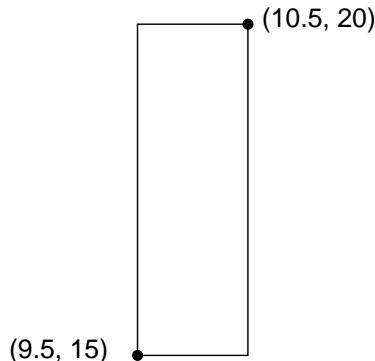
<b>Keywords</b>	<b>Displayed Results</b>
RAREA, RLENGTH, RWIDTH	Area of contact and via resistors, length and width for all other extracted resistors.
RAREA, RLENGTH, RWIDTH, RLAYER	Area of contact and via resistors, length and width for all other extracted resistors, and layers for all extracted resistors.

Specify the RVIACOUNT option if you want the netlist to include the via count for each extracted resistor. This information is useful when performing EM analysis.

Specify the TCOEFF option if you want the netlist to include temperature coefficients. If there is no reduction, the TC1 and TC2 from the appropriate [Resistance Sheet](#) statement are netlisted. This requires that the TC1 and TC2 values be added to the Resistance Sheet statement. If the parasitics are reduced using [PEX Reduce Digital](#) or [PEX Reduce ROnly](#), then TC1 and TC2 are calculated for each reduced element. Parasitic resistors will list the resistance value at the nominal temperature and TC1 and TC2. In the simulation tool, set the nominal temperature to the nominal temperature in the [PEX Thickness EQN](#) statement.

When using RLOCATION with the ADITRAIL option to generate a DSPF netlist, the Calibre xRC tool generates x,y locations based on polygon information instead of centerline.

[Figure 4-290](#) illustrates XY locations as the corners of a rectangle rather than the center line.

**Figure 4-290. ADITRAIL xRC netlist output example**

```
R1 A B 10 $w=1 $l=5 $X1=9.5 $Y1=15 $X2=10.5 $Y2=20
```

## Using SCHEMATICONLY

The **SCHEMATICONLY** parameter generates a parasitic netlist that contains the devices and nets in the source netlist used for LVS. The device parameters are extracted from the layout and backannotated to the cross-referenced device in the source netlist.

All extracted device parameters are netlisted. Transistor-level and gate-level extractions are supported. Full hierarchical extraction (-xcell -full) is not supported.

Often the hierarchy between the source and the layout does not match. A one-to-many device mapping is quite likely. In such cases, as well as for one-to-one device mapping, you can annotate parameter values from the layout into the source. This can be done when you want to use the pre-layout simulation test bench to verify your design. This also helps to increase the accuracy of your results.

When using the **SCHEMATICONLY** with the *filename*, the Calibre xRC tool calculates the parameters

- Length (L)
- Width (W)
- Drain/Source area (AD and AS)
- Drain/Source perimeter (PD and PS)

for MOS devices from the layout and adds them to the netlist. These calculations use the equations shown in [Table 4-43](#) and [Table 4-44](#).

With PEX Netlist ... SCHEMATICONLY, only standard MOSFET parameters are netlisted. Other device elements are netlisted with the source parameters. You can use the *filename* option to specify any element and its parameter settings.

**Table 4-43. PEX Netlist: Parameter Calculations for Parallel Transistors**

Source/Layout Mapping	Width Calculation	Length Calculation	Area/Perimeter Calculation
<b>1:N</b> - 1 transistor in the source maps to N parallel-connected transistors in the layout	$W_x = \text{Sum}(W_1, W_2, \dots, W_N)$	$L_x = \text{Min}(L_1, L_2, \dots, L_N)$	$AD_x = \text{Sum}(AD_1, AD_2, \dots, AD_N)$ $PD_x = \text{Sum}(PD_1, PD_2, \dots, PD_N)$ Calculations for AS and PS are the same.
<b>M:1</b> - M parallel-connected transistors in the source map to 1 transistor in the layout	$W_x = W/M$	$L_x = L$	$AD_x = AD/M$ $PD_x = PD/M$ Calculations for AS and PS are the same.
<b>M:M</b> - There is a 1-to-1 mapping between parallel-connected transistors in the source and layout	$W_{x1} = W_1$ $W_{x2} = W_2$ $\dots$ $W_{xM} = W_M$	$L_{x1} = L_1$ $L_{x2} = L_1$ $\dots$ $L_{xM} = L_M$	$AD_{x1} = AD_1, AD_{x2} = AD_2, \dots, AD_{xM} = AD_M$ $PD_{x1} = PD_1, PD_{x2} = PD_2, \dots, PD_{xM} = PD_M$ Calculations for AS and PS are the same.
<b>M:N</b> - M transistors in the source map to N transistors in the layout	$W_{x[1..M]} = \text{Sum}(W_1, W_2, \dots, W_N)/M$	$L_{x[1..M]} = \text{Min}(L_1, L_2, \dots, L_N)$	$AD_{[1..M]} = \text{Sum}(AD_1, AD_2, \dots, AD_N) / M$ $PD_{[1..M]} = \text{Sum}(PD_1, PD_2, \dots, PD_N) / M$ Calculations for AS and PS are the same.

**Table 4-44. PEX Netlist: Parameter Calculations for Series Transistors**

Source/Layout Mapping	Width Calculation	Length Calculation	Area/Periphery Calculation
<b>1:N</b> - 1 transistor in the source maps to N series-connected transistors in the layout	$W_x = \text{Sum}(W_1, W_2 \dots W_N) / N$	$L_x = \text{Sum}(L_1, L_2 \dots L_N)$	$AD_x = \text{Sum}(AD_1, AD_2 \dots AD_N)$ $PD_x = \text{Sum}(PD_1, PD_2 \dots PD_N)$ Calculations for AS and PS are the same.
<b>M:1</b> - M series-connected transistors in the source map to 1 transistor in the layout	$W_{x[1..M]} = W$	$L_{x[1..M]} = L / M$	$AD_{x[1..M]} = AD / M$ $PD_{x[1..M]} = PD / M$ Calculations for AS and PS are the same.
<b>M:M</b> - There is a 1-to-1 mapping between series-connected transistors in the source and layout	$W_{x1} = W_1$ $W_{x2} = W_2$ $\dots$ $W_{xM} = W_M$	$L_{x1} = L_1$ $L_{x2} = L_2$ $\dots$ $L_{xM} = L_M$	$AD_{x1} = AD_1 \dots AD_{xM} = AD_M$ $PD_{x1} = PD_1 \dots PD_{xM} = PD_M$ Calculations for AS and PS are the same.

## Examples

### Example 1

The following example outputs a netlist named netlist.spc in HSPICE format. It sets the scale to 1e-6 meters so parameters are entered in microns and areas are entered in square microns; obtains net and instance names from the layout; sets VSS as ground; and reports device locations as comments:

```
PEX NETLIST "netlist.spc" HSPICE 1e-06 LAYOUTNAMES GROUND VSS LOCATION
```

### Example 2

The following example outputs a netlist named netlist1 in HSPICE format. Net and instance names are obtained from the source and changes a net named /x14/x13/x12 to x14\_x13\_x12:

```
PEX NETLIST "netlist.spc" HSPICE SOURCENAMES SEPARATOR "_"
```

### Example 3

The following example outputs a netlist named netlist2 in HSPICE format. Net and instance names are obtained from the layout and net names are shortened. For example a net named /inst1/inst2/inst3/inst4/inst45 is changed to 45. The modified netlist can be simulated by HSPICE.

```
PEX NETLIST "netlist.spc" HSPICE LAYOUTNAMES SHORTPINNAMES
```

### Example 4

The following example outputs the temperature coefficients when no reduction is performed.

```
RESISTANCE SHEET metall [1.29 0 0.00512 0.00013]
PEX NETLIST "netlist.spc" HSPICE SOURCENAMES TCOEFF
```

With the lines above, if parasitic resistor r26 is from metal1, then the output netlist in HSPICE format will look like:

```
r26 228 230 0.115235 tc1=0.00512 tc2=0.00013
```

#### **Example 5**

The following example outputs a single netlist file named netlist.hspc in HSPICE format. It combines the transistor-level HSPICE file with the .pex and .pxi files into one output netlist file:

```
PEX NETLIST "netlist.hspc" HSPICE 1 SOURCEBASED SINGLEFILE
```

This single netlist file will include the contents of the transistor-level HSPICE netlist, the parasitic net models for each extracted net (subcircuit definitions) including parasitic capacitors and resistors, and the instances of the parasitic models. The .include lines that normally appear in the HSPICE netlist will be commented out when using the SINGLEFILE option.

#### **Example 6**

The following example outputs the area, length, and width values for extracted resistors in a DSPF netlist based on the source netlist. Intrinsic capacitors with the VSS net designated as the grounding net are also extracted.

```
PEX NETLIST "netlist.dspf" DSPF SOURCENAMES RLENGTH RWIDTH RAREA RLAYER
GROUND VSS
```

The resulting output is:

```
cGND/11 GND:14 VSS 18.2156f
cGND/12 GND:8 VSS 6.68235f
cGND/13 GND:6 VSS 28.8203f
cGND/14 GND:2 VSS 30.0372f
rGND/15 MZNTG:b GND:16 16.1807 $w=1.5e-07 $l=2.38296e-07 $layer=tpdiff
rGND/16 GND:14 GND:16 26 $a=8.1e-15 $layer=odCont
rGND/17 GND:14 GND:18 0.103422 $w=1.66e-07 $l=0.12e-07 $layer=metall1
```

To render the resistor property values for width, length, and area in units of microns, include the RSCALE option with a value of 1e-06.

```
PEX NETLIST "netlist.dspf" DSPF SOURCENAMES RSCALE 1e-06 RLENGTH RWIDTH
RAREA RLAYER GROUND VSS
```

The resulting output is:

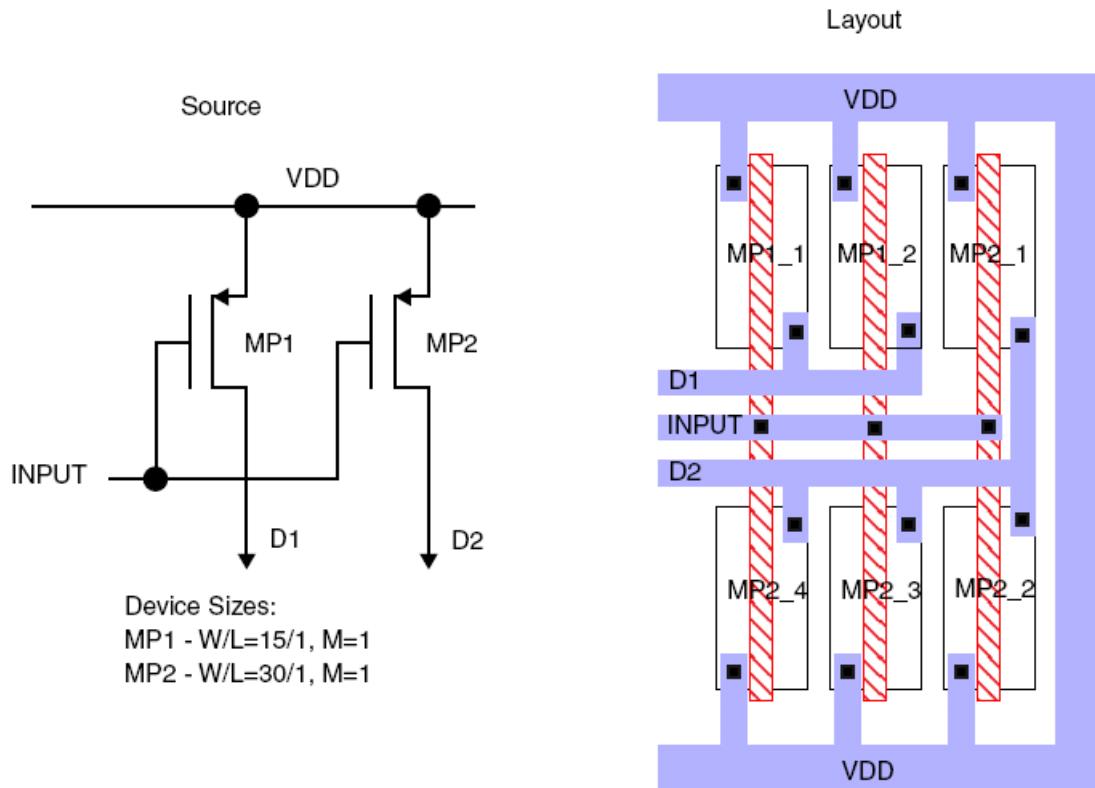
```
cGND/11 GND:14 VSS 18.2156f
cGND/12 GND:8 VSS 6.68235f
cGND/13 GND:6 VSS 28.8203f
cGND/14 GND:2 VSS 30.0372f
rGND/15 MZNTG:b GND:16 16.1807 $w=0.15 $l=0.238296 $layer=tpdiff
rGND/16 GND:14 GND:16 26 $a=0.0081 $layer=odCont
rGND/17 GND:14 GND:18 0.103422 $w=0.166 $l=0.012 $layer=metall1
```

**Example 7**

This example shows how a one-to-many mapping of transistors between the schematic and layout is handled. Figure 4-292 shows two PMOS transistors with parallel gate and source connections. In the layout, MP1 is rendered in two equally sized transistors, 7.5/1. MP2 is rendered in four equally sized transistors with the same size. The drain/source widths for both transistors are assumed to be 1. The SVRF statement is:

```
PEX NETLIST "netlist_ba.spc" HSPICE SCHEMATICONLY
```

**Figure 4-291. Example of 1:N Transistor Mapping**



The resulting backannotated HSPICE netlist for these two transistors is:

```
XMMMP1 D1 INPUT VDD VDD L=1 W=15 AD=15 AS=15 PD=34 PS=34
XMMMP2 D2 INPUT VDD VDD L=1 W=30 AD=30 AS=30 PD=68 PS=68
```

**Example 8**

The following example outputs a netlist based on the schematic with resistors backannotated with width and length from the layout:

```
PEX NETLIST "netlist.spc" HSPICE SCHEMATICONLY "myDevices"
```

The file myDevices contains the following definition:

```
// RP1: Poly1 Resistor model parameters
ELEMENT R MODEL R1
```

```
[  
    L = max //use maximum L value  
    W = max //use maximum W value  
]
```

**Example 9**

The following example outputs the via count for each parasitic resistor as a comment in the extracted netlist:

```
PEX NETLIST "netlist.spc" HSPICE SOURCenames RWidth RLAYER RVIACOUNT
```

The following is a sample from the resulting output netlist in HSPICE format:

```
r26 12 13 18.76 $w=9e-08 $layer=polyCont $count=2
```

Where \$count represents the number of vias for parasitic resistor r26.

**Example 10**

The following example outputs the subtype name for the device subtype in the extracted CalibreView netlist:

```
PEX NETLIST "netlist.cv" CALIBREVIEW SOURCenames ICELLMAP RWidth RLAYER
```

The following is a sample from the resulting output netlist in CALIBREVIEW format:

```
mr_pi "PMOS" "MM1" `("MM1_d" "MM1_g" "vdd!" "vdd!")' (( "l" 4e-06) ("w"  
1.2e-05) ("m" 1) ("lpe" 0) ("lvs_model" "PMOS1")) '(38.85 23.48)
```

Where ("lvs\_model" "PMOS1") represents the property containing the subtype name.

# PEX Netlist ADMS

Parasitic extraction

## PEX NETLIST ADMS *filename*

```

{ HSPICE [scale]
| DSPF [scale] [PRIMETIME | HSIM]
| SPEF [PRIMETIME]
| SPECTRE [scale]
| ELDO [scale]}
{LAYOUTNAMES
| SOURCENAMES}
[GROUND ground_name] [SEPARATOR sep_name]
{BLOCK blockname}
[NOXANNOTATE | XANNOTATE]
[NOFILTERX | FILTERX]
[MODE {C | R | RC | RCC}]]
```

## Summary

Generates a netlist for the extracted block in an ADMS flow.

## Parameters

- *filename*

A required filename for the created netlist.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

If the *filename* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. For HSPICE and ELDO formats, each file is individually compressed. The references in the main netlist to the other files includes the suffix. This functionality assumes the compress and gzip commands are available in your \$PATH.

- One of the following must be specified. Possible choices are:

**HSPICE** [*scale*] — The netlist format is HSPICE.

**DSPF** [*scale*] [PRIMETIME | HSIM] — The netlist format is DSPF.

**SPEF** [PRIMETIME] — The netlist format is SPEF.

**SPECTRE** [*scale*] — The netlist format is Spectre.

**ELDO** [*scale*] — The netlist format is Eldo.

The optional string *scale* must be a positive floating-point number and may be a variable. It specifies the size multiplier (in units of meters) for the L, W, A, P, AS, AD, PS, and PD parameters of MOSFET, diode, and JFET devices. Set *scale* to 1e-6 meters to enter the parameters in microns; areas are then in square microns. Default is 1.

The optional keyword PRIMETIME creates a more compact netlist that the PrimeTime® static timing analysis tool will be able to parse. This keyword is only supported in DSPF and SPEF formats. When PRIMETIME is specified, Xs are removed in front of device names and keywords and element names are in all uppercase. Additionally, in DSPF, lumped capacitance values are placed in the NET section and the Instance section is not written out.

- One of the following must be specified. Possible choices are:

**LAYOUTNAMES** — Specifies the names are derived from the layout. May not be specified with **SOURCENAMES**.

**SOURCENAMES** — Specifies the names are derived from the schematic. When specifying the **SOURCENAMES** option, make sure that a valid [Source Primary](#) statement identifying a valid primary cell is in the rule file.

- **GROUND** *ground\_name*

An optional keyword set, where *ground\_name* is the name of the ground node, that specifies the ground node. All capacitances are grounded to this node. The default value of *ground\_name* is 0.

- **SEPARATOR** *sep\_name*

An optional keyword set, where *sep\_name* allows you to replace the default hierarchy separator slash (/) with *sep\_name* in any net or instance name from the source or layout. If the first character in the name is a slash, it is deleted. Note that for Eldo netlist format the default hierarchy separator is underscore (\_).

- **BLOCK** *blockname*

An optional keyword that specifies a name for the block. You do not have to specify **BLOCK** *blockname* as part of the rule, as long as the top-level cell is the PRIMARY cell for the design (SOURCENAMES or LAYOUTNAMES) based on the naming convention specified in the PEX Netlist ADMS statement.

- **NOXANNOTATE** | **XANNOTATE**

An optional keyword that specifies whether to perform cross-annotation on a block. The default is **NOXANNOTATE**, which means cross-annotation is not performed. You can enable cross-annotation for any block independently of any other block by specifying **XANNOTATE** on any specific block's PEX Netlist ADMS specification statement.

- **NOFILTERX** | **FILTERX**

An optional keyword that causes the formatter to filter X's in HSPICE, DSPF, SPEF, and Spectre output. Turn on filtering by including FILTERX in the PEX Netlist ADMS statement in your rule file. The default is **NOFILTERX**, which means filtering is turned off.

- MODE {C | R | RC | RCC}

An optional keyword that allows you to specify the mode to use when extracting the block. Possible choices are:

- C — C only extraction.
- R — R only extraction.
- RC — Distributed RC extraction.
- RCC — R coupled-C extraction.

## Description

Generates a netlist for the extracted block in an ADMS flow.

For each block in an ADMS flow that is on a domain boundary, there must exist a PEX Netlist ADMS rule in the rule file.

## Examples

```
PEX NETLIST ADMS "top_analog.rcc.spf" DSPF SOURCENAMES GROUND VSS  
SEPARATOR ":" BLOCK "top_analog" MODE RCC
```

```
PEX NETLIST ADMS "digital.rc.spef" SPEF SOURCENAMES GROUND VSS  
SEPARATOR ":" BLOCK "digital" MODE RC FILTERX
```

# PEX Netlist Capacitance Unit

Parasitic extraction

## PEX NETLIST CAPACITANCE UNIT *unit*

### Parameters

- *unit*

A required unit of capacitance for parasitic netlists and reports. Unit may be given in SI prefix form (such as pf, nf, or ff) or equivalent SI number form (such as 1e-06, 1e-09, or 1e-15). Extraction stops with an error message if this value is missing or invalid.

### Description

Allows a common unit of capacitance to apply to all parasitic netlists and reports. Use this statement to control the capacitance units for DSPF, SPEF, HSPICE, ELDO, CalibreView, and Spectre netlist formats. This statement is optional and may only be specified once.

If you do not include this statement in your rule file, the default unit of capacitance used depends on the netlist format. The default unit of capacitance for the SPEF netlist format is picofarads. All other netlist formats auto scale values to an appropriate unit, which is typically femtofarads.

### Examples

To set capacitance units used by all netlist formats to picofarads, include one of the following statements in your rule file:

```
PEX NETLIST CAPACITANCE UNIT pf
```

or

```
PEX NETLIST CAPACITANCE UNIT 1e-12
```

# PEX Netlist Character Map

Parasitic Extraction

**PEX NETLIST CHARACTER MAP {NONE | “oldchar newchar [oldchar newchar]...”}**

## Parameters

- **NONE**

Required keyword if character mappings are not specified. **NONE** disables character replacement. This is the default behavior if the statement is not in the rule file.

- ***oldchar newchar***

Required parameters that specify to substitute ***newchar*** for ***oldchar*** in net names. You can specify any number of additional pairs. Embedded spaces and unpaired characters are ignored. The entire string of pairs must be enclosed in double quotes.

## Description

Replaces characters that are part of net names, including parasitic models. It does not affect other types of information.

### Text Changed:

Net names

Instance names

Model names

### Text Not Changed:

Property specifications

Separators indicating hierarchy in net names

Netlist keywords

### Note



Separator characters are not replaced. If the design has been flattened into an xcell then the name for a net that is part of the flattened hierarchy reflects the former hierarchy for that net. If you need to change the hierarchical indicator, use the SEPARATOR keyword in the [PEX Netlist](#) statement to define the character as shown in [Example 2](#).

If this statement is specified along with PEX Netlist Smashed\_Device Delimiter and PEX Netlist Replicated\_Device Delimiter, it does not affect the device delimiters. If those statements are not specified, it does.

## Examples

### Example 1

The following example replaces equal signs (=) with underscores (\_):

```
PEX NETLIST CHARACTER MAP "=_"
```

Note that property specifications, such as w=1e-06, are *not* changed.

**Example 2**

The following example replaces equal signs (=) and non-hierarchical slashes (/) with underscores (\_):

```
PEX NETLIST CHARACTER MAP "=_ /_"
```

The space between the first underscore and the slash is not required, but makes it easier to read.

This character map acts on the net name portions of a flattened hierarchy, but not the separator character. For example, if a net named W/R is part of a flattened hierarchy, such that its full net name is cell1/cell2/cell3/W/R, then the slashes between cells will not be replaced but the slash in W/R will. This produces cell1/cell2/cell3/W\_R as the final name.

**Example 3**

This example turns off all character replacement:

```
PEX NETLIST CHARACTER MAP NONE
```

## PEX Netlist Connection Section

Parasitic extraction

### PEX NETLIST CONNECTION SECTION {YES | NO}

#### Parameters

- **YES**

Required keyword that indicates nonstandard SPICE statements will be included in the connection section of the netlist.

- **NO**

Required keyword that indicates nonstandard SPICE statements will not be included in the connection section of the netlist.

#### Description

This statement specifies whether or not to include the statements beginning with ‘\*|I’ or ‘\*|P’ in a DSPF or SPEF netlist. If PEX Netlist Connection Section is not included in a rule file, then statements beginning with ‘\*|I’ or ‘\*|P’ occur in the connection section.

#### Examples

This statement will include netlist code with ‘\*|I’ or ‘\*|P’ at the beginning of a line:

```
PEX NETLIST CONNECTION SECTION YES
```

## PEX Netlist Create Smashed Device Names

Parasitic extraction

**PEX NETLIST CREATE SMASHED DEVICE NAMES {YES | NO}**

### Parameters

- **YES**

Specifies to create new instance names for smashed devices. This is the default.

- **NO**

Specifies to leave smashed device instance names unchanged.

### Description

Specifies how to output smashed device instance names in DSPF and SPEF formats. The default is to create new instance names for smashed devices.

### Examples

Include the following statement in your rule file if you do not want new instance names for smashed devices:

```
PEX NETLIST CREATE SMASHED DEVICE NAMES NO
```

## PEX Netlist Device Resistance Model

Parasitic extraction

### PEX NETLIST DEVICE RESISTANCE MODEL {SPICE | **CALCULATE**}

#### Parameters

- **SPICE**

Specifies using the internal HSPICE model for parasitic resistance calculation. This is the default.

- **CALCULATE**

Specifies using the width (W) and length (L) from the DEVICE statement in the LVS rule file for parasitic resistance calculation.

#### Description

Specifies which model to use for parasitic resistors. You can specify whether to use the internal calculations from the HSPICE model or measured W and L as calculated by a DEVICE specification statement. For more information, see the Device SVRF statement.

#### Examples

For an HSPICE netlist, include the following statement in your rule file to calculate parasitic resistance using the W and L from the DEVICE statement:

```
PEX NETLIST DEVICE RESISTANCE MODEL CALCULATE
```

# PEX Netlist Distributed

Parasitic extraction

**Note**



As of the 2008.4 release, this statement has been deprecated. Use [PEX Netlist](#) instead.

## PEX NETLIST DISTRIBUTED NONE

### PEX NETLIST DISTRIBUTED *filename*

```
{ HSPICE [scale]
| DSPF [scale] [PRIMETIME]
| SPEF [PRIMETIME]
| SPECTRE [scale]
| ELDO [scale]
| CALIBREVIEW [scale] }
{ID | LAYOUT | SOURCE}
[GROUND ground_name]
[PRUNE]
[RSCALE value]
[SEPARATOR sep_name]
[LOCATION]
[RLOCATION]
[RAREA]
[RWIDTH] [RLENGTH]
[RTHICKNESS]
[RLAYER]
[RCNAMED]
[TCOEFF]
```

## Summary

Generates a netlist for the extracted circuit with distributed parasitic elements and places it in the file specified.

## Parameters

- **NONE**

Keyword that specifies that no distributed netlist is to be generated.

- ***filename***

A required filename for the created netlist.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

If the *filename* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. For HSPICE and ELDO formats, each file is individually compressed. The references in the main netlist to the other files includes the suffix. This functionality assumes the compress and gzip commands are available in your \$PATH.

- One of the following must be specified. Possible choices are:

**HSPICE** [*scale*] — The netlist format is HSPICE.

**DSPF** [*scale*] [PRIMETIME] — The netlist format is DSPF.

**SPEF** [PRIMETIME] — The netlist format is SPEF.

**SPECTRE** [*scale*] — The netlist format is Spectre.

**ELDO** [*scale*] — The netlist format is Eldo.

**CALIBREVIEW** [*scale*] — The netlist format is CalibreView.

The optional string *scale* must be a positive floating-point number and may be a variable. It specifies the size multiplier (in units of meters) for the L, W, A, P, AS, AD, PS, and PD parameters of MOSFET, diode, and JFET devices. Set *scale* to 1e-6 meters to enter the parameters in microns; areas are then in square microns. Default is 1.

The optional keyword PRIMETIME creates a more compact netlist that the PrimeTime® static timing analysis tool will be able to parse. This keyword is only supported in DSPF and SPEF formats in RC mode. When PRIMETIME is specified, Xs are removed in front of device names and keywords and element names are in all uppercase. Additionally, in DSPF, lumped capacitance values are placed in the NET section and the Instance section is not written out.

- One of the following must be specified. Possible choices are:

**ID** — Specifies the names are derived from layout handles.

**LAYOUT** — Specifies the names are derived from the layout.

**SOURCE** — Specifies the names are derived from the schematic or netlist. When specifying the **SOURCE** option, make sure that a valid [Source Primary](#) statement identifying a valid primary cell is in the rule file.

- **GROUND** *ground\_name*

An optional keyword set, where *ground\_name* is the name of the ground node, that specifies the ground node. All capacitances are grounded to this node. The default value of *ground\_name* is 0.

- **PRUNE**

An optional keyword that removes all intentional devices not attached to the extracted nets. Use this keyword with selected net extraction as described in “[Netlisting Only Direct Devices on a Selected Net](#)” in the [Calibre xRC User’s Guide](#).

- **RSCALE *value***

An optional keyword set, where *value* is a scaling factor that changes parasitic resistor properties, such as width, independently of device properties. The default value is 1.

- **SEPARATOR *sep\_name***

An optional keyword set, where *sep\_name* allows you to replace the default hierarchy separator slash (/) with *sep\_name* in any net or instance name from the source or layout. If the first character in the name is a slash, it is deleted. Note that for Eldo netlist format the default hierarchy separator is underscore (\_).

- **LOCATION**

An optional keyword that specifies to report device locations as comments. It supports the locations for intentional resistors and capacitors.

- **RLOCATION**

An optional keyword that specifies to report parasitic resistor locations as comments. For CalibreView format netlists, it also reports parasitic capacitor locations.

- **RAREA**

An optional keyword that specifies to report contact/via area used for extracting parasitic resistors. The value is reported as a comment. RAREA is only supported with the DSPF format.

- **RWIDTH**

An optional keyword that specifies to report parasitic resistor widths as comments.

- **RLENGTH**

An optional keyword that specifies to report parasitic resistor lengths as comments.

- **RTHICKNESS**

An optional keyword that specifies to report calculated parasitic resistor thicknesses as comments. When resistance is specified with [Resistance Sheet](#), thickness is not reported.

- **RLAYER**

An optional keyword that specifies to report parasitic resistor layers as comments.

- **RCNAMED**

An optional keyword that causes the R or C value of an intentional device to be replaced with a parameter (“R=*value*” or “C=*value*”) in HSPICE, DSPF, or Eldo netlists for R or C devices that have models.

- **TCOEFF**

An optional keyword that specifies to output temperature coefficients for parasitic resistors for the DSPF, Eldo, HSPICE, and SPECTRE formats. It is ignored for CalibreView and SPEF.

## Description

Generates a netlist for the extracted circuit with distributed parasitic elements and placed in the file specified.

Specify the RWIDTH, RLENGTH, RAREA, RLAYER, and RLOCATION options if you want Calibre xRC to include width, length, via/contact area, layer data, and location information in your parasitic databases. These cause PDBs and parasitic netlists to be larger.

### Note



If reduction removes a resistor, Calibre xRC may generate invalid area, location, width, length, and layer information.

In DSPF and HSPICE, the output netlist results are:

- RWIDTH: \$w=*number*
- RLENGTH: \$l=*number*
- RAREA: \$a=*number*
- RLAYER: \$layer=*layer\_name*
- RLOCATION: \$x=*x\_coord* \$y=*y\_coord* \$x2=*x2\_coord* \$y2=*y2\_coord*

The x and y coordinates correspond to the x and y coordinates of the nodes to which the parasitic resistor is connected. You must specify the keywords at the time you invoke `calibre -xrc -pdb` to store the area, location, width, length, and layer information in the PDB.

Several keyword combinations are possible for displaying physical properties of parasitic devices. [Table 4-45](#) shows the possible combinations and the resulting outputs.

**Table 4-45. PEX Netlist Distributed Keyword Combinations and Results**

Keywords	Displayed Results
RAREA	Only area of contact and via resistors is displayed.
RAREA, RWIDTH	Area of contact and via resistors and width of other extracted resistors.
RAREA, RLENGTH	Area of contact and via resistors and length of other extracted resistors.
RAREA, RLAYER	Area of contact and via resistors and layers for all extracted resistors.
RAREA, RWIDTH, RLAYER	Area of contact and via resistors, width for other extracted resistors, layers for all extracted resistors.
RAREA, RLENGTH, RLAYER	Area of contact and via resistors, length for other extracted resistors, layers for all extracted resistors.
RAREA, RLENGTH, RWIDTH	Area of contact and via resistors, length and width for all other extracted resistors.

**Table 4-45. PEX Netlist Distributed Keyword Combinations and Results**

<b>Keywords</b>	<b>Displayed Results</b>
RAREA, RLENGTH, RWIDTH, RLAYER	Area of contact and via resistors, length and width for all other extracted resistors, and layers for all extracted resistors.
	Specify the TCOEFF option if you want the netlist to include temperature coefficients. If there is no reduction, the TC1 and TC2 from the appropriate <a href="#">Resistance Sheet</a> statement are netlisted. This requires that the TC1 and TC2 values be added to the Resistance Sheet statement. If the parasitics are reduced using <a href="#">PEX Reduce Digital</a> or <a href="#">PEX Reduce ROnly</a> , then TC1 and TC2 are calculated for each reduced element. Parasitic resistors will list the resistance value at the nominal temperature and TC1 and TC2. In the simulation tool, set the nominal temperature to the nominal temperature in the <a href="#">PEX Thickness EQN</a> statement.

## Examples

### Example 1

The following example outputs a distributed netlist named netlist.dist in HSPICE format. It sets the scale to 1e-6 meters so parameters are entered in microns and areas are entered in square microns; obtains net and instance names from the layout; sets VSS as ground; and reports device locations as comments:

```
PEX NETLIST DISTRIBUTED "netlist.dist" HSPICE 1e-6 LAYOUT GROUND VSS
LOCATION
```

### Example 2

The following example outputs a distributed netlist named netlist1 in HSPICE format. Net and instance names are obtained from the source and changes a net named /x14/x13/x12 to x14\_x13\_x12:

```
PEX NETLIST DISTRIBUTED "netlist1" HSPICE SOURCE SEPARATOR "_"
```

### Example 3

The following example outputs a distributed netlist named netlist2 in HSPICE format. Net and instance names are obtained from the layout and changes a net named /inst1/inst2/inst3/inst4/inst45 to 45. The modified netlist can be simulated by HSPICE.

```
PEX NETLIST DISTRIBUTED "netlist2" HSPICE LAYOUT
```

### Example 4

The following example outputs the temperature coefficients when no reduction is performed.

```
RESISTANCE SHEET metal1 [1.29 0 0.00512 0.00013]
PEX NETLIST DISTRIBUTED "netlist1" HSPICE SOURCE TCOEFF
```

With the lines above, if parasitic resistor r26 is from metal1, then the output netlist in HSPICE format will look like:

```
r26 228 230 0.115235 tc1=0.00512 tc2=0.00013
```

**Example 5**

The following example outputs the area, length, and width values for extracted resistors in a DSPF netlist based on the source netlist. Intrinsic capacitors with the VSS net designated as the grounding net are also extracted.

```
PEX NETLIST DISTRIBUTED "netlist.dspf" DSPF SOURCE RLENGTH RWIDTH RAREA  
RLAYER GROUND VSS
```

The resulting output is:

```
cGND/11 GND:14 VSS 18.2156f  
cGND/12 GND:8 VSS 6.68235f  
cGND/13 GND:6 VSS 28.8203f  
cGND/14 GND:2 VSS 30.0372f  
rGND/15 MZNTG:b GND:16 16.1807 $w=1.5e-07 $l=2.38296 $layer=tpdiff  
rGND/16 GND:14 GND:16 26 $a=8.1e-15 $layer=odCont  
rGND/17 GND:14 GND:18 0.103422 $w=1.66e-07 $l=0.12 $layer=metall1
```

# PEX Netlist Escape Characters

Parasitic extraction

## PEX NETLIST ESCAPE CHARACTERS *string*

### Parameters

- *string*

Specifies escape character strings used in the output file. The default value is !@#\$%^&\*(){}\\:;`<>?/.,-=.

### Description

Specifies escape characters that can be used in the output file. You must enclose the string within quotation marks.

This statement may be useful for modifying how escape characters are handled in different netlist formats.

### Examples

To restore the set of default escape characters, include the following statement in your rule file:

```
PEX NETLIST ESCAPE CHARACTERS " !@#$%^&*(){}\\:;`<>?/.,-= "
```

## PEX Netlist Export Ports

Parasitic extraction

### PEX NETLIST EXPORT PORTS {NO | YES}

#### Parameters

- **NO**

Required keyword that instructs the formatter not to create ports for internal nets. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Required keyword that instructs the formatter to create ports for internal nets.

#### Description

This statement controls whether or not the formatter creates ports for internal nets and includes them in the extracted netlist. These ports allow you to probe or plot internal nets during the simulation step.

This statement may be specified once in your rule file.

#### Examples

To create ports for internal nets and write them to the extracted netlist, specify the following statement in your rule file:

```
PEX NETLIST EXPORT PORTS YES
```

# PEX Netlist Filter

Parasitic extraction

**PEX NETLIST FILTER** *component\_type* [(*component\_subtype*)]  
[LAYOUT | SOURCE] [SHORT]

## Parameters

- ***component\_type***  
A required component type for the filter. The tool removes all instances of this component from the generated netlist. The ***component\_type*** may be any SPICE device type or primitive subcircuit name. Wildcards are not permitted.
- (*component\_subtype*)  
An optional component subtype for the filter. This option must be specified in parentheses. The filter ignores subtype when you do not specify this parameter.
- **LAYOUT**  
An optional keyword used for filtering devices exclusively from the layout-based netlist. This is the default. If LAYOUT is specified and you are producing a source-based netlist, the statement has no effect.
- **SOURCE**  
An optional keyword used for filtering devices exclusively from the source-based netlist. If SOURCE is specified and you are producing a layout-based netlist, the statement has no effect.
- **SHORT**  
An optional keyword which causes the formatter to short the nets connected to the filtered device.

## Description

Filters out device instances during the formatter stage based upon component type in either layout or source. This statement is typically used to filter SPICE device elements. This statement is compatible with all netlist formats. The filtering applies only during netlist formatting. Specifying both LAYOUT and SOURCE generates an error.

If SHORT is specified, then the nets that connect to the device are shorted together. When a device is filtered and shorted, at least one of the device pins must be on a net attached to a port.

## Examples

### Example 1

This example filters a capacitor from a source-based netlist:

```
X27/M15 NETA NETB NETC VDD PCH L=6e-08 W=1.7e-07 M=1
C1 NETX NETY CP 1000f
```

## PEX Netlist Filter

---

Include the following statement in the rule file to filter out the specific intentional capacitor C(CP) from the output netlist:

```
PEX NETLIST FILTER C(CP) SOURCE
```

This results in the following:

```
X27/M15 NETA NETB NETC VDD PCH L=6e-08 W=1.7e-07 M=1
```

### Example 2

This example filters all the U\_RES devices from a layout-based netlist:

```
*|I (X0:BOTTOM X0 BOTTOM B 0.0)
*|I (X1:BOTTOM X1 BOTTOM B 0.0)
c3/5 X0:BOTTOM 0 17.6736f
r3/10 3:31 X0:BOTTOM 0.0494685 $layer=V3
r3/18 X1:BOTTOM X0:BOTTOM 0.01
cc_6 S_GND X0:BOTTOM 9.51043p

* Instance *
X0 A1 X0:BOTTOM U_RES
X1 A2 X1:BOTTOM U_RES
X2 X2:TOP B C4_RES
```

Include the following statement in the rule file to remove the U\_RES devices and resolve their connections as shorts:

```
PEX NETLIST FILTER U_RES SHORT
```

This results in the following:

```
*|P (A1 X 0.0)
*|P (A2 X 0.0)
c3/5 A1 0 17.6736f
r3/10 3:31 A1 0.0494685 $layer=V3
r3/18 A2 A1 0.01
cc_6 S_GND A1 9.51043p

* Instance *
X2 X2:TOP B C4_RES
```

## PEX Netlist Global Nets

Parasitic extraction

**PEX NETLIST GLOBAL NETS** *name* [*name* ...]

### Parameters

- *name*

Specifies a net to be removed from the subcircuit and places it in a global name definition. More than one name can be specified.

### Description

Removes the named nets from a subcircuit and places them in a global name definition.

Using this statement with global power or ground nets causes the formatter stage to remove the named nets from the pin list of the instantiated subcircuit and places them in a global name definition, making the output netlist more compact.

Global nets must also be listed in the [PEX Extract Exclude](#) statement. You cannot netlist parasitics on global nets in hierarchical extraction.

### Examples

To place the VSS and VDD nets in a global net list, include the following statement in the rule file:

```
PEX NETLIST GLOBAL NETS VSS VDD
```

## PEX Netlist Linewrap

Parasitic extraction

### PEX NETLIST LINEWRAP {NO | *value*}

#### Parameters

- **NO**  
Required keyword that instructs the netlist formatter to not use linewrapping.
- ***value***  
Required keyword that instructs the netlist formatter to wrap netlist text at a given value.  
The ***value*** must be greater than or equal to 80. The default value is 80.

#### Description

This statement specifies where the netlist formatter wraps the output line in the generated netlist. If value is less than 80 then the following error message is generated:

```
Error PEX59 on line NN of filename - improper numeric value (must be an integer and >= 80): value
```

#### Examples

##### Example 1

This statement instructs the netlist formatter to wrap text at column 100:

```
PEX NETLIST LINEWRAP 100
```

##### Example 2

By default the netlist formatter wraps at column 80. The following is an example from a SPICE netlist:

```
mXI7/XI1/MPO N_INB13_XI7/XI1/MPO_d N_INB03_XI7/XI1/MPO_g  
+ N_vdd!_XI7/XI1/MPO_s N_vdd!_XI7/XI5/MPO_b P L=2.4e-07 W=1e-06 M=1
```

Include the following statement in your rule file if you do not want lines greater than 80 characters to wrap:

```
PEX NETLIST LINEWRAP NO
```

The output will be on one line and will not wrap:

```
mXI7/XI1/MPO N_INB13_XI7/XI1/MPO_d N_INB03_XI7/XI1/MPO_g N_vdd!_XI7/ ...
```

## PEX Netlist LPE Ignore Idealnet

Parasitic extraction

### PEX NETLIST LPE IGNORE IDEALNET {YES | NO}

#### Parameters

- **YES**

Required keyword that instructs the netlist formatter to ignore nets that were excluded from extraction when performing LPE calculations.

- **NO**

Required keyword that instructs the netlist formatter to consider nets that were excluded from extraction when calculating LPE parameter values. This is the default behavior when you do not include this statement in the rule file.

#### Description

This statement specifies whether or not the netlist formatter takes into account the nets that have been excluded from extraction when determining the worst case parasitic model. For more information on LPE parameters and syntax, see the [PEX BA Mapfile](#) statement.

#### Examples

This statement will ignore nets that have been excluded from extraction when calculating LPE parameter values:

```
PEX NETLIST LPE IGNORE IDEALNET YES
```

## PEX Netlist LPE Using Extmode

Parasitic extraction

### PEX NETLIST LPE USING EXTMODE{YES | NO}

#### Parameters

- **YES**

Required keyword that instructs the netlist formatter to obtain LPE values from the mapfile based on extraction mode, such as -r, -c, -rc, -rcc, and so forth.

- **NO**

Required keyword that instructs the netlist formatter to obtain LPE values using the net parasitic models. This is the default behavior when you do not include this statement in the rule file.

#### Description

This statement instructs the netlist formatter to obtain LPE values from the mapfile based on the command line extraction mode, such as -r, -c, -rc, -rcc, and so forth. If the extraction mode is -rcc, then RC LPE values are used from the mapfile.

#### Examples

##### Example 1

To control which LPE values are used based on the PDB extraction mode, include the following statements in your rule file:

```
PEX NETLIST LPE USING EXTMODE YES  
PEX BA MAPFILE mymapfile
```

The LPE values written to the netlist are obtained from the mapfile based on extraction mode. For example, if the following LPE exist in *mymapfile*:

```
%Models  
N N LPE foo NORC 0 C 100 R 200 RC 300  
P P LPE NORC 0 C 1 R 2 RC 3
```

And extraction mode is -r, then the values used for resistance will be 200 for models named foo and 2 for models with no name.

##### Example 2

If your netlist format is CALIBREVIEW and you want to use the LPE values from the mapfile, include the following statements in your rule file:

```
PEX NETLIST LPE USING EXTMODE YES  
PEX BA MAPFILE mymapfile CALIBREVIEW
```

# PEX Netlist Lumped

Parasitic extraction

**Note**



As of the 2008.4 release, this statement has been deprecated. Use [PEX Netlist](#) instead.

## PEX NETLIST LUMPED NONE

### PEX NETLIST LUMPED *filename*

```
{ HSPICE [scale]
| DSPF [scale]
| SPEF
| SPECTRE [scale]
| ELDO [scale]
| CALIBREVIEW [scale]}
{ID | LAYOUT | SOURCE}
[GROUND ground_name] [SEPARATOR sep_name] [LOCATION] [RCNAMED]
```

## Summary

Generates a netlist for the extracted circuit with lumped parasitic elements and places it in the file specified.

## Parameters

- **NONE**

A required keyword that specifies that no lumped netlist is to be generated.

- ***filename***

A required filename for the created netlist.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

If the ***filename*** ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. For HSPICE and ELDO formats, each file is individually compressed. The references in the main netlist to the other files includes the suffix. This functionality assumes the compress and gzip commands are available in your \$PATH.

- One of the following must be specified. Possible choices are:

**HSPICE [scale]** — The netlist format is HSPICE.

**DSPF [scale]** — The netlist format is DSPF.

**SPEF** — The netlist format is SPEF.

**SPECTRE [scale]** — The netlist format is Spectre.

**ELDO** [*scale*] — The netlist format is Eldo.

**CALIBREVIEW** [*scale*] — The netlist format is CalibreView.

The optional *scale* must be a positive floating-point number and may be a numeric variable. It specifies the size multiplier (in units of meters) for the L, W, A, P, AS, AD, PS, and PD parameters of MOSFET, diode, and JFET devices. Set *scale* to 1e-6 meters to enter the parameters in microns; areas are then in square microns. Default is 1.

- One of the following must be specified. Possible choices are:

**ID** — Specifies the names are derived from layout handles.

**LAYOUT** — Specifies the names are derived from the layout.

**SOURCE** — Specifies the names are derived from the schematic. When specifying the **SOURCE** option, make sure that a valid [Source Primary](#) statement identifying a valid primary cell is in the rule file.

- **GROUND** *ground\_name*

An optional keyword, where *ground\_name* is the name of the ground node, that specifies the ground node. All capacitances are grounded to this node. The default value of *ground\_node* is 0.

- **SEPARATOR** *sep\_name*

An optional keyword, where *sep\_name* allows you to replace the default hierarchy separator slash (/) with *sep\_name* in any net or instance name from the source or layout. If the first character in the name is a /, it will be deleted. Note that for Eldo netlist format the default hierarchy separator is underscore (\_).

- **LOCATION**

An optional keyword that specifies to report device locations as comments. It supports the locations for intentional resistors and capacitors.

- **RCNAMED**

An optional keyword that causes the R or C value of an intentional device to be replaced with a parameter (“R=*value*” or “C=*value*”) in HSPICE or Eldo netlists for R or C devices that have models.

### Description

Generates a netlist for the extracted circuit with lumped parasitic elements and placed in the file specified.

## Examples

### Example 1

The following example outputs a lumped netlist named netlist.lumped in HSPICE format. It sets the scale to 1e-6 meters so parameters are entered in microns and areas are entered in square microns; obtains net and instance names from the layout; sets VSS as ground; and reports device locations as comments:

```
PEX NETLIST LUMPED "netlist.lumped" HSPICE 1e-6 LAYOUT GROUND VSS LOCATION
```

### Example 2

The following example outputs a lumped netlist named netlist1 in HSPICE format. Net and instance names are obtained from the source and changes a net named /x14/x13/x12 to x14\_x13\_x12:

```
PEX NETLIST LUMPED "netlist1" HSPICE SOURCE SEPARATOR "_"
```

### Example 3

The following example outputs a lumped netlist named netlist2 in HSPICE format. Net and instance names are obtained from the layout and changes a net named /inst1/inst2/inst3/inst4/inst45 to 45. The modified netlist can now be simulated by HSPICE, which has a 16-character maximum name length constraint.

```
PEX NETLIST LUMPED "netlist2" HSPICE LAYOUT
```

## PEX Netlist Mutual Resistance

Parasitic extraction

### PEX NETLIST MUTUAL RESISTANCE {YES | NO}

Used only in Calibre xL.

#### Parameters

- **YES**  
Specifies to use current-controlled voltage sources to model mutual resistance. This is the default setting.
- **NO**  
Specifies to disable using current-controlled voltage sources to model mutual resistance.

#### Description

Specifies whether the Calibre xL tool uses current-controlled voltage sources to model mutual resistance. When you specify **NO**, the Calibre xL tool produces a parasitic netlist without current-controlled voltage sources. Use this setting if your simulator does not support current-controlled voltage sources in a file. The default is **YES**, which produces a parasitic netlist with current-controlled voltage sources.

#### Examples

To turn off using voltage-controlled voltage sources, use the following:

```
PEX NETLIST MUTUAL RESISTANCE NO
```

# PEX Netlist Noxref Net Names

Parasitic extraction

## PEX NETLIST NOXREF NET NAMES {YES | NO}

### Parameters

- **YES**

Required parameter that specifies keeping parasitic net models marked with \_noxref. Nets in the design with “noxref” in the name still appear in the intentional circuit and are treated as ideal nets. **YES** is the default.

- **NO**

Required parameter that specifies suppressing parasitic net models from the netlist for \_noxref nets.

### Description

Keeps or eliminates parasitic net models marked with \_noxref from netlists. For more information on \_noxref nets, see “[Interpreting \\_noxref Entries](#)” in the *Calibre xRC User’s Manual*.

### Examples

For a design with one intentional resistor in the schematic, but two in series in the layout, using the PEX Netlist Noxref Net Names statement affects the output of the simple HSPICE netlist as shown below:

#### NOXREF = YES

```
% grep -n -i subckt ./simple.rcc.pex
8:.subckt PM_SIMPLE%X1/4 1 6 8 9 12
14 15 17
26:.subckt PM_SIMPLE%X2/4 1 6 8 9 12
14 15 17
44:.subckt PM_SIMPLE%A 3 5
50:.subckt PM_SIMPLE%noxref_2 2 3
56:.subckt PM_SIMPLE%B 2 5
61:.subckt PM_SIMPLE%VDD 1 4 9 10 12
    14 16
75:.subckt PM_SIMPLE%4 1 2 4 11
83:.subckt PM_SIMPLE%GND 1 4 12 13
93:.subckt PM_SIMPLE%6 1 2 4 11
```

#### NOXREF = NO

```
% grep -n -i subckt ./simple.rcc.pex
8:.subckt PM_SIMPLE%X1/4 1 6 8 9 12
14 15 17
26:.subckt PM_SIMPLE%X2/4 1 6 8 9 12
14 15 17
44:.subckt PM_SIMPLE%A 3 5
50:.subckt PM_SIMPLE%B 2 5
55:.subckt PM_SIMPLE%VDD 1 4 9 10 12
    14 16
69:.subckt PM_SIMPLE%4 1 2 4 11
77:.subckt PM_SIMPLE%GND 1 4 12 13
87:.subckt PM_SIMPLE%6 1 2 4 11
```

The subcircuit in line 50 of the NOXREF = YES column does not appear in the NOXREF = NO column because it is suppressed. This subcircuit is a parasitic net model for the noxref\_2 net as shown by the name “PM\_SIMPLE%noxref\_2”.

# PEX Netlist Position File

Parasitic extraction

**PEX NETLIST POSITION FILE {NO | YES | *filename*}**

## Parameters

- **NO**  
Keyword indicating to not write the file. This is the default if this statement is not specified.
- **YES**  
Keyword indicating to write the position file. The file is named *top\_cell.placement\_info* where *top\_cell* is derived from the [Layout Primary](#) statement.
- ***filename***  
Parameter specifying the filename to use instead of the default.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

## Description

Generates a hierarchical instance position file for use with DSPF netlists in the HSIM-RA simulator. The file contains the orientation and placement information of cell instances.

Only cell instances are written to the position file. User-defined devices are not written, nor remapped instances. The file follows the syntax described in the *HSIMplus User’s Manual*, version Y-2006.06.

When the netlist format is not DSPF, the file is not created.

## Examples

The following lines cause a netlist position file named *primary.placement\_info* to be created when you run the formatter:

```
LAYOUT PRIMARY primary
PEX NETLIST DSPF "netlist.dspf" LAYOUTNAMES
PEX NETLIST POSITION FILE YES
```

# PEX Netlist Replicated\_Device Delimiter

Parasitic extraction

## PEX NETLIST REPLICATED\_DEVICE DELIMITER *character*

### Parameters

- *character*

Required string argument specifying the character used when constructing unique identifiers for devices that occur once in the source but multiple times in the layout. If you specify more than one character in the string, only the first character is used. The default if the statement is not specified is underscore (\_).

### Description

Specifies the delimiting character for replicated layout instances in the netlist. The Calibre nmLVS SPICE output uses two equal signs, ==, followed by an instance number. This is not valid for many simulators, so the Calibre xRC formatter uses underscores instead. This statement allows you to specify your own character.

If this SVRF statement does not exist in the rule file, then the default value of \_ is used to separate the device name from the instance number appended.

If PEX Netlist Replicated\_Device Delimiter is used then the [PEX Netlist Character Map](#) statement does not transform the characters.

### Examples

A layout may have multiple instances of device M1. The Calibre nmLVS SPICE netlist would show these as follows:

```
M1
M1==1
M1==2
```

and so forth.

A SPICE netlist produced by the Calibre xRC formatter would represent the same devices as:

```
M1
M1__1
M1__2
```

and so forth.

The following line defines a replicated device name delimiting string used in the extracted netlist:

```
PEX NETLIST REPLICATED_DEVICE DELIMITER ' | '
```

The result is as follows:

```
M1  
M1 | | 1  
M1 | | 2
```

and so forth.

# PEX Netlist SchematicOnly

Parasitic extraction

**Note**



As of the 2008.4 release, this statement has been deprecated. Use the SCHEMATICONLY parameter in the [PEX Netlist](#) SVRF statement instead.

## PEX NETLIST SCHEMATICONLY {NO | YES | *filename*}

### Parameters

- **NO**  
Switches off generation of a parasitic netlist, based on the LVS source hierarchy, with backannotated device parameters extracted from the layout. NO is default when the statement is not included in a rule file.
- **YES**  
Generates a parasitic netlist based on the LVS source hierarchy, with intentional device parameters extracted and backannotated from the layout. Parasitics are netlisted only for nets that are cross-referenced to the source netlist.
- ***filename***  
Required name of a file containing entries in the following format for defining device parameters:

```
ELEMENT element_name MODEL model_name
[
  prop_type1 = val1
  prop_type2 = val2
  ...
  prop_typeN = valN
]
```

where:

- ELEMENT is a required keyword, correlated to the device statements from LVS.
- *element\_name* is the LVS standard device name, such as R for resistor, C for capacitor, and so forth. [Table 4-5](#) shows a complete list of available default devices.
- MODEL is an optional keyword. If this keyword is missing from any element, then the same calculation will be applied to all elements without MODEL specified.
- *model\_name* is the device model name, such as RP for poly resistor, RN for n-well resistor, and so forth.
- *prop\_type* is the parameter that will be calculated for the device.

- o *valN* can be set to:

min	use minimum value from layout
max	use maximum value from layout
median	use median value from layout
sum	add layout values for parameter and divide by source device count
ave	add layout values for parameter and divide by layout device count
source	use value from source

The input file may contain comments. Comments begin with // and run to the end of the line.  
The square brackets are not optional and must be included.

## Description

Generates a parasitic netlist that contains the devices and nets in the source netlist used for LVS. The device parameters are extracted from the layout and backannotated to the cross-referenced device in the source netlist.

All extracted device parameters are netlisted. Transistor-level and gate-level extractions are supported. Full hierarchical extraction (-xcell -full) is not supported. All netlist formats are supported.

Often the hierarchy between the source and the layout does not match. A one-to-many device mapping is quite likely. In such cases, you can annotate parameter values from the layout into the source. This can be done when you want to use the pre-layout simulation test bench to verify your design. This also helps to increase the accuracy of your results.

When using the PEX Netlist SchematicOnly YES, the Calibre xRC tool calculates the parameters

- Length (L)
- Width (W)
- Drain/Source area (AD and AS)
- Drain/Source perimeter (PD and PS)

for MOS devices from the layout and annotates them in your output netlist. These calculations use the equations shown in [Table 4-46](#) and [Table 4-47](#).

With PEX Netlist SchematicOnly YES, only SPICE parameters for standard MOS transistors are netlisted. Other device elements are netlisted with the source parameters. You can use the *filename* option to specify any element and its parameter settings.

**Table 4-46. PEX Netlist SchematicOnly: Parameter Calculations for Parallel Transistors**

Source/Layout Mapping	Width Calculation	Length Calculation	Area/Perimeter Calculation
<b>1:N</b> - 1 transistor in the source maps to N parallel-connected transistors in the layout	$W_x = \text{Sum}(W_1, W_2, \dots, W_N)$	$L_x = \text{Min}(L_1, L_2, \dots, L_N)$	$AD_x = \text{Sum}(AD_1, AD_2, \dots, AD_N)$ $PD_x = \text{Sum}(PD_1, PD_2, \dots, PD_N)$ Calculations for AS and PS are the same.
<b>M:1</b> - M parallel-connected transistors in the source map to 1 transistor in the layout	$W_x = W/M$	$L_x = L$	$AD_x = AD/M$ $PD_x = PD/M$ Calculations for AS and PS are the same.
<b>M:M</b> - There is a 1-to-1 mapping between parallel-connected transistors in the source and layout	$W_{x1} = W_1$ $W_{x2} = W_2$ ... $W_{xM} = W_M$	$L_{x1} = L_1$ $L_{x2} = L_1$ ... $L_{xM} = L_M$	$AD_{x1} = AD_1, AD_{x2} = AD_2, \dots, AD_{xM} = AD_M$ $PD_{x1} = PD_1, PD_{x2} = PD_2, \dots, PD_{xM} = PD_M$ Calculations for AS and PS are the same.
<b>M:N</b> - M transistors in the source map to N transistors in the layout	$W_{x[1..M]} = \text{Sum}(W_1, W_2, \dots, W_N)/M$	$L_{x[1..M]} = \text{Min}(L_1, L_2, \dots, L_N)$	$AD_{[1..M]} = \text{Sum}(AD_1, AD_2, \dots, AD_N) / M$ $PD_{[1..M]} = \text{Sum}(PD_1, PD_2, \dots, PD_N) / M$ Calculations for AS and PS are the same.

**Table 4-47. PEX Netlist SchematicOnly: Parameter Calculations for Series Transistors**

Source/Layout Mapping	Width Calculation	Length Calculation	Area/Periphery Calculation
<b>1:N</b> - 1 transistor in the source maps to N series-connected transistors in the layout	$W_x = \text{Sum}(W_1, W_2, \dots, W_N) / N$	$L_x = \text{Sum}(L_1, L_2, \dots, L_N)$	$AD_x = \text{Sum}(AD_1, AD_2, \dots, AD_N)$ $PD_x = \text{Sum}(PD_1, PD_2, \dots, PD_N)$ Calculations for AS and PS are the same.
<b>M:1</b> - M series-connected transistors in the source map to 1 transistor in the layout	$W_{x[1..M]} = W$	$L_{x[1..M]} = L / M$	$AD_{x[1..M]} = AD / M$ $PD_{x[1..M]} = PD / M$ Calculations for AS and PS are the same.

**Table 4-47. PEX Netlist SchematicOnly: Parameter Calculations for Series Transistors**

Source/Layout Mapping	Width Calculation	Length Calculation	Area/Periphery Calculation
M:M - There is a 1-to-1 mapping between series-connected transistors in the source and layout	$W_{x1} = W_1$ $W_{x2} = W_2$ ... $W_{xM} = W_M$	$L_{x1} = L_1$ $L_{x2} = L_2$ ... $L_{xM} = W_M$	$AD_{x1} = AD_1 \dots AD_{xM} = AD_M$ $PD_{x1} = PD_1 \dots PD_{xM} = PD_M$ Calculations for AS and PS are the same.

## Examples

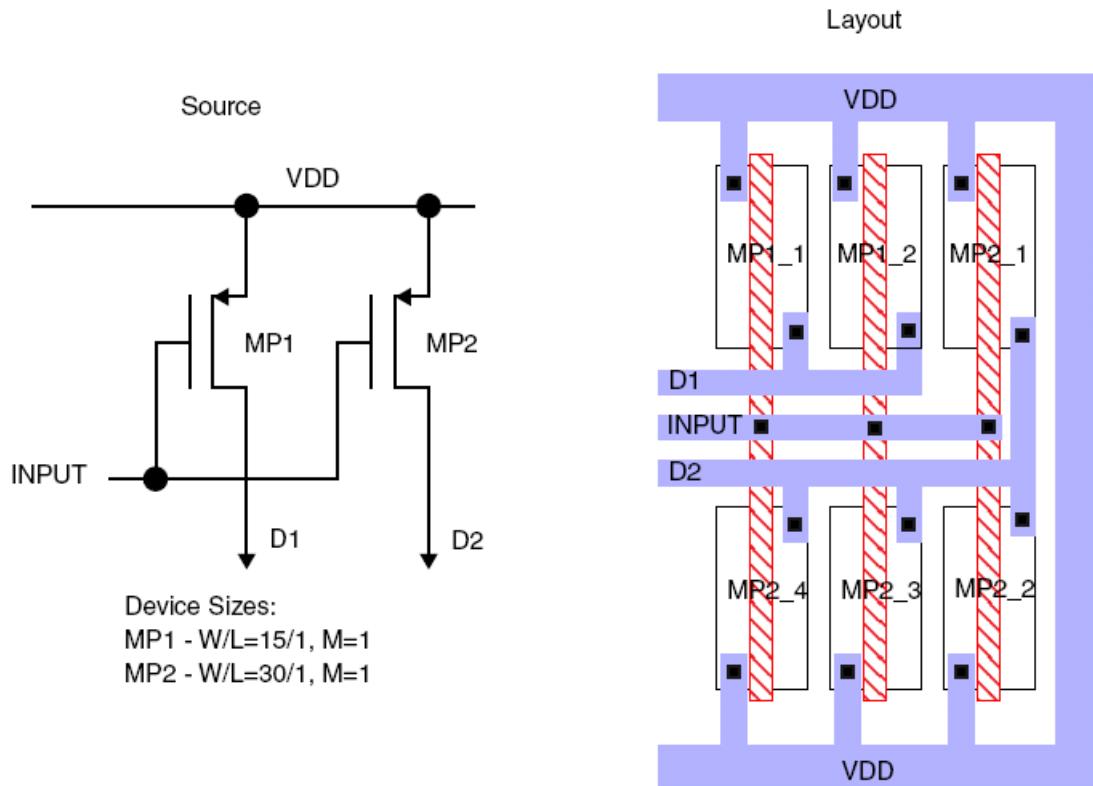
### Example 1

In this example, you can see how a one-to-many mapping of transistors between the schematic and layout is handled. The following statement can be included in your rule file:

```
PEX NETLIST SCHEMATICONLY YES
```

Figure 4-292 shows two PMOS transistors with parallel gate and source connections. In the layout, MP1 is rendered in two equally sized transistors, 7.5/1. MP2 is rendered in four equally sized transistors with the same size. The drain/source widths for both transistors are assumed to be 1.

**Figure 4-292. Example of 1:N Transistor Mapping**



The resulting backannotated HSPICE netlist for these two transistors is:

```
XMMP1 D1 INPUT VDD VDD L=1 W=15 AD=15 AS=15 PD=34 PS=34
XMMP2 D2 INPUT VDD VDD L=1 W=30 AD=30 AS=30 PD=68 PS=68
```

### Example 2

You can include the following statement in your rule file if you want to specify specific devices:

```
PEX NETLIST SCHEMATICONLY "./myDevices"
```

The file *myDevices* contains the following definition:

```
// RP1: Poly1 Resistor model parameters
ELEMENT R MODEL R1
[
    L = max //use maximum L value
    W = max //use maximum W value
]
```

## PEX Netlist Select File

Parasitic extraction

### PEX NETLIST SELECT FILE *netfile*

#### Summary

Specifies a file which is used for controlling the extracted netlist during the formatting stage.

#### Parameters

- *netfile*

Specifies a file that is used to assign parasitic models on a net-by-net basis during the formatting stage. The file has the following format:

```
net_name      net_model
```

where:

**net\_name** A net from the design. Net names are case sensitive. Net names can contain hierarchy, for example:

```
XI1/XI5/SIGNAL_A
```

Net names can also contain a wildcard, for example:

```
XI1/XI5/SIGNAL?
```

The hierarchical path for net name cannot contain a wildcard. The following is invalid:

```
XI1/XI?/SIGNAL_A
```

The first line in *netfile* can be used to specify a default net\_model for all net names that are not listed in the netfile. This is specified as follows:

```
=default=      net_model
```

- `net_model` The model used for the net during netlist formatting. Net models are not case sensitive. The options are:
- **Ideal** — Includes only the intentional devices on the net. This provides the least detailed net representation.
  - **R** — Includes resistors in the netlist.
  - **Cg** — Includes grounded intrinsic and coupling capacitors in the netlist.
  - **C** — Includes intrinsic and coupling capacitors.
  - **RC** — Includes a distributed network of resistors and capacitors in the netlist. Intrinsic and coupled capacitors are grounded.
  - **RCC** — Includes a distributed network of resistors and capacitors in the netlist.
  - **RL** — Includes a distributed network of resistors and inductors.
  - **RLM** — Includes a distributed network of resistors and inductors along with mutual inductance in the netlist.
  - **RCL** — Includes a distributed network of resistors, inductors, and capacitors in the netlist.
  - **RCLM** — Includes a distributed network of resistors, inductors, and capacitors along with mutual inductance in the netlist.
  - **RCCL** — Includes a distributed network of resistors, inductors, and capacitors in the netlist. Coupled capacitors are not grounded.
  - **RCCLM** — Includes a distributed network of resistors, inductors, and capacitors along with mutual inductance in the netlist. Coupled capacitors are not grounded.
  - **L** — Includes a distributed network of inductors in the netlist.
  - **LM** — Includes a distributed network of inductors along with mutual inductance in the netlist.

## Description

Specifies a file containing net names and net models used for formatting the extracted netlist. Use this SVRF statement to define mapping for a specific net model to a selected net. You must provide hierarchical LVS results to use this statement.

The net model is applied during the formatting stage of extraction (-fmt). By specifying the PEX Netlist Select File statement in the rule file, you can generate multiple parasitic netlists by changing the net models in the file without rerunning the parasitic extraction step (-pdb).

The following are pre-requisites for using PEX Netlist Select File:

- Run LVS hierarchically (-hier)
- Run the PDB generation stage with either -rcc, -rccl, or -rcclm (for example, -pdb -rcc)

- Run without the -select switch
- Run extraction for your design in a single run; running with multiple PDBs is not allowed
- Run the formatter with the -netmodel command line option, as follows:

```
calibre -xrc -fmt -netmodel ...
```

If -netmodel is specified in the formatter invocation and a PEX Netlist Select File statement does not exist in the rule file, or there are errors in the netfile, the Calibre xRC tool issues a warning.

For more information about PDB extraction and formatter invocation switches, see “[calibre -xrc -pdb](#)” and “[calibre -xrc -fmt](#)” in the *Calibre xRC User’s Manual*.

## Net Name Handling

Net names specified in *netfile* must correspond to the nets specified by the option selected in **PEX Netlist** statement. For example, if specifying PEX Netlist ... LAYOUTNAMES in the rule file, then use layout net names in *netfile*. If a specified net is not found in the design, then the Calibre xRC tool issues a warning.

Use “=default=” and a net\_model to specify a default model for all nets in the design that are not specified in *netfile*. This must be included as the first line of the file, for example:

```
=default=          Cg  
XI1/XI2/SIGNAL_A    RC  
...
```

In this example, all the nets that are not named in the net file will be included in the output with the Cg net model.

If a net is specified in the net file with one entry, the entire net is assigned the most detailed net model. For example:

```
=default=          R  
XI1/NET_B          RC
```

In this example, the default for all unnamed nets is R. However, NET\_B in instance XI1 and all nets in the design hierarchy that attach to this net are modeled with RC, the most detailed model.

If =default= is not specified in the net file, then all unnamed nets are modeled with the RCC net model. This is a parasitic extraction mode that can be used with PEX Netlist Select File (-pdb -rcc).

Nets can be specified at any level of hierarchy. They can be specified at the top-level of the design and also cell-level by using the full instance path. For example, to include all of the nets inside the instance XI2, use a wildcard and the full instance path in the following way:

```
BLOCK_A/XI1/XI2/?      RC
```

This definition states that all of the nets in instance XI2 will be extracted using the RC net model. That is, all of the nets in XI2 will be represented with distributed models that include resistors, grounded intrinsic capacitors, and grounded coupled capacitors.

A net may be specified with multiple net\_name entries in the net file. For example:

```
NET_A      R          //connects NET_B and NET_C at top level
XI1/NET_B  RCC
XI2/NET_C  IDEAL    //XI 2 is a pcell instance
```

In this example, NET\_A connects together nets XI1/NET\_B and XI2/NET\_C at a higher level in the design hierarchy. The Calibre xRC tool emits a conflict resolution warning and assigns the most detail net model to the entire net, RCC. The only exception is the net segment inside the Pcell, XI2/NET\_C. This will remain as an ideal net (no parasitics).

## Supported Options

All Calibre xRC netlist formats are supported. These include CalibreView, Eldo, DSPF, HSPICE, PRIMETIME, and SPEF. Layer aliasing, netlist reduction, and other netlisting methods may be applied to output generated with the statement.

The following extraction methods are supported:

- Transistor-level — Produces a flat, transistor-level netlist. Use -pdb [-rcc | -rccl | -rcclm].
- Gate-level — Produces a hierarchical, gate-level netlist. Use -pdb [-rcc | -rccl | -rcclm] -xcell.

## Unsupported Options

The following extraction methods and options are not supported:

- Full hierarchical (-xcell -full), ADMS (-adms), and In-Context (-xcell -incontext) extraction or formatting.
- Using ASIC mode during the extraction stage, -pdb -asic.
- Incremental parasitic extraction, found in Calibre Interactive.
- Calibre xRC CB, -cb.
- Using the -xcell or -hcell command line options during the formatting stage, -fmt -xcell or -hcell.
- The PRUNE netlisting option, [PEX Netlist ... PRUNE](#).

## Examples

The following example shows how to apply PEX Netlist Select File for extracting parasitics for nets in a design. [Figure 4-293](#) represents a hierarchical view of a design with several cells and nets. Cell contents are shown with as schematic representation while the interconnect is shown with a layout representation.

## PEX Netlist Select File

To specify the file which contains a list of nets and corresponding net models, include the following statement in your rule file:

```
PEX NETLIST SELECT FILE designNets.list
```

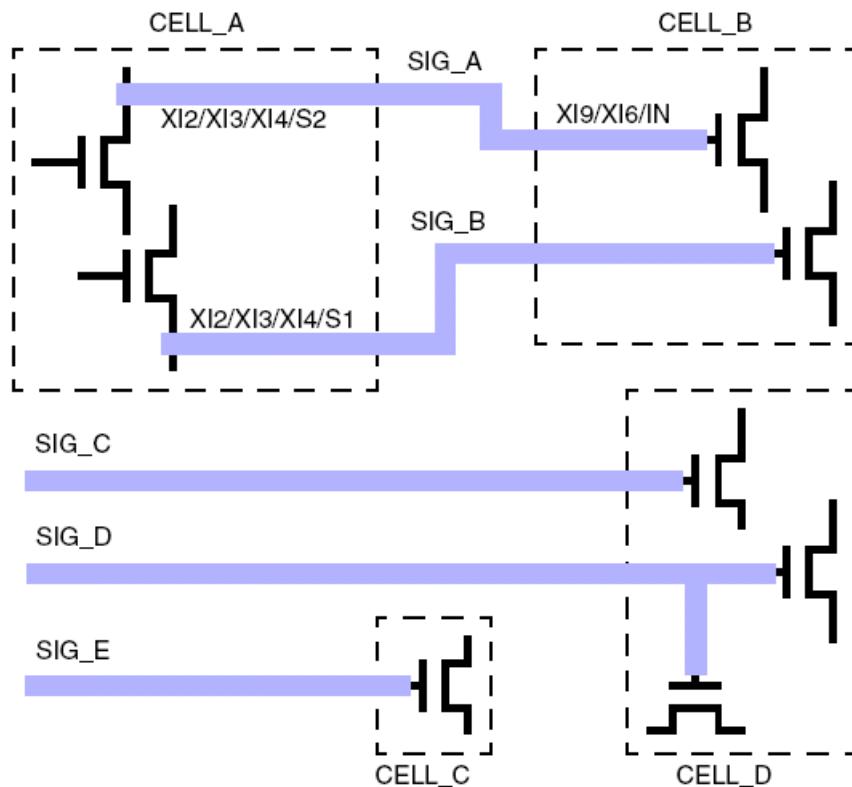
When using PEX Netlist Select File, invoke the Calibre xRC tool as follows:

```
calibre -lvs -hier -spice my_design_dir/myDesign.sp myRules  
calibre -xrc -pdb -rcc myRules  
calibre -xrc -fmt -netmodel myRules
```

The netfile designNets.list contains the following nets and corresponding models:

```
=default=      C  
SIG_A         RC  
VDD          IDEAL  
XI2/XI3/XI4/? C      //nets in CELL_A  
XI9/XI6/IN   IDEAL    //net in CELL_B  
XI1/VDD      RC  
SIG_D         RCC     //parallel to SIG_E, connects to CELL_D  
SIG_E         C       //attaches to CELL_C, a Pcell instance
```

**Figure 4-293. Hierarchical Design and Interconnect**



The resulting netlist is illustrated in [Figure 4-294](#). The effective net model for each net is also shown. Nets VDD and XI1/VDD are not shown.

The default net\_model for the design is C. All nets which are not listed in designNets.list will be formatted with this net model. The capacitors for each net segment include the intrinsic

capacitance for the segment plus the coupling capacitance to adjacent nets lumped with the intrinsic capacitance.

#### **SIG\_A Net**

All nets in CELL\_A are specified with the C net model. The net XI9/XI6/IN in CELL\_B is specified as IDEAL. Net XI2/XI3/XI4/S2 and XI9/XI6/IN are connected together with SIG\_A at a higher level in the hierarchy. The net model for SIG\_A is RC. Since the RC net model provides greater detail than either IDEAL or C net models, the net is netlisted as SIG\_A with the RC net model.

#### **SIG\_B Net**

Net XI2/XI3/XI4/S2 is specified with the C net model. SIG\_B is not listed in the netfile. The result is that the entire net is netlisted as SIG\_B with the C net model.

#### **SIG\_C Net**

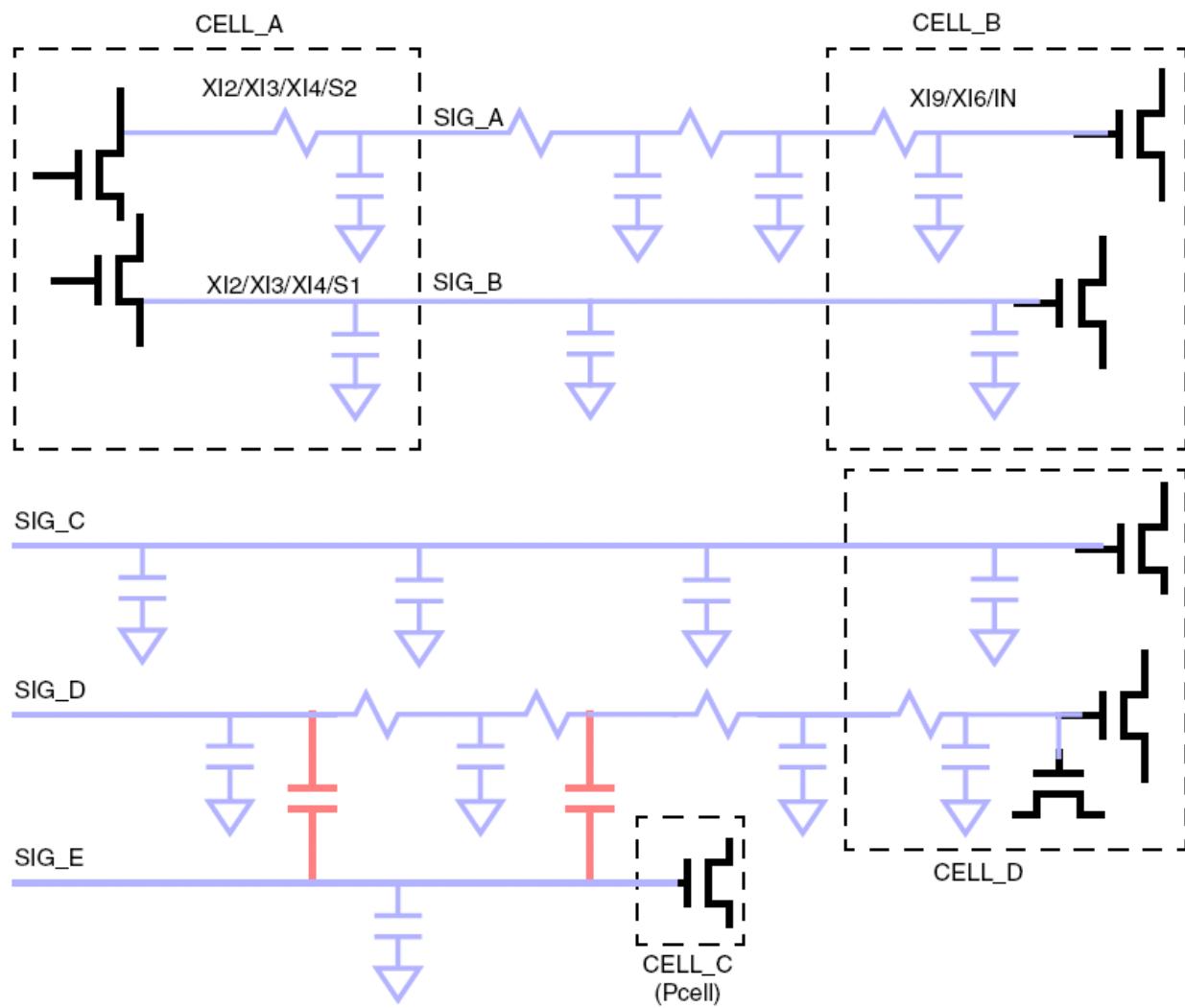
SIG\_C uses the default net model, C. Every net attached to SIG\_C will use this model since these nets are not explicitly listed in the net file.

#### **SIG\_D Net**

SIG\_D is specified with the RCC net model. Since SIG\_C uses the default C net model, the coupling caps from SIG\_D to SIG\_C have been lumped together with the intrinsic capacitances on SIG\_D.

#### **SIG\_E Net**

SIG\_E is modeled with the C net model. Intrinsic capacitors for SIG\_E and coupling capacitors to SIG\_D are included in the netlist. The internal nets of CELL\_C, a Pcell, do not use the default net model. The nets for the Pcell will not include any parasitics.

**Figure 4-294. Net Model Representation of Design**

# PEX Netlist Simple

Parasitic extraction

```
PEX NETLIST SIMPLE filename { CALIBREVIEW [scale] | ELDO [scale]
| HSPICE [scale] | SPECTRE [scale]} {LAYOUTNAMES | SOURCENAMES}
[SEPARATOR sep_name] [LOCATION] [RCNAMED]
```

## Summary

Generates a netlist for the extracted circuit without parasitic elements and places it in the specified file.

## Parameters

- ***filename***

A required filename for the created netlist.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

If the ***filename*** ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. For the HSPICE format, each file is individually compressed. The references in the main netlist to the other files includes the suffix. This functionality assumes the compress and gzip commands are available in your \$PATH.

- One of the following must be specified. Possible choices are:

**CALIBREVIEW** *scale* — The netlist format is CalibreView.

**ELDO** *scale* — The netlist format is ELDO.

**HSPICE** *scale* — The netlist format is HSPICE.

**SPECTRE** *scale* — The netlist format is Spectre.

The ***scale*** must be a positive, floating-point number and may be a numeric variable. It specifies the size multiplier (in units of meters) for the L, W, A, P, AS, AD, PS, and PD parameters of MOSFET, diode, and JFET devices. Set ***scale*** to 1e-6 meters to enter the parameters in microns; areas are then in square microns. Default is 1.

- One of the following must be specified. Possible choices are:

**LAYOUTNAMES** — Specifies the names are derived from the layout.

**SOURCENAMES** — Specifies the names are derived from the schematic. When specifying the **SOURCENAMES** option, make sure that a valid [Source Primary](#) statement identifying a valid primary cell is in the rule file.

- **SEPARATOR** *sep\_name*

Keyword set, where ***sep\_name*** allows you to replace the default hierarchy separator slash (/) with ***sep\_name*** in any net or instance name from the source or layout. If the first character in

## PEX Netlist Simple

---

the name is a /, it is deleted. Note that for Eldo netlist format the default hierarchy separator is underscore (\_).

- LOCATION

An optional keyword that specifies to report device locations as comments. It supports the locations for intentional resistors and capacitors.

- RCNAMED

An optional keyword that causes the R or C value of an intentional device to be replaced with a parameter (“R=*value*” or “C=*value*”) in HSPICE netlists for R or C devices that have models.

### Description

Generates a netlist for the extracted circuit *without parasitic elements* and places it in the specified file.

### Examples

#### Example 1

The following example outputs a simple netlist named netlist.simple in HSPICE format. Net and instance names are obtained from the layout and device locations are reported as comments.

```
PEX NETLIST SIMPLE "netlist.simple" HSPICE LAYOUTNAMES LOCATION
```

#### Example 2

The following example outputs a simple netlist named “netlist1” in HSPICE format. Net and instance names are obtained from the source and changes a net named /x14/x13/x12 to x14\_x13\_x12:

```
PEX NETLIST SIMPLE "netlist1" HSPICE SOURCENAMES SEPARATOR "_"
```

# PEX Netlist Smashed\_Device Delimiter

Parasitic extraction

## **PEX NETLIST SMASHED\_DEVICE DELIMITER *symbol***

### Parameters

- *symbol*

Required string argument specifying the character or characters used when constructing unique identifiers for devices that are a single device in the source but multiple parts in the layout, also known as smashed or flattened devices. The default if the statement is not specified is @.

### Description

Specifies the delimiting character for smashed, or flattened, layout instances in the layout netlist.

If this SVRF statement does not exist in the rule file, then the default value of @ is used to separate the device name from the appended instance number.

If PEX Netlist Smashed\_Device Delimiter is used then the [PEX Netlist Character Map](#) statement does not transform the characters.

PEX Netlist Smashed\_Device Delimiter is applied only to smashed devices and takes precedence over the PEX Character Map statement, which applies to all of the devices in a netlist.

### Examples

The following line defines a smashed device name delimiting character used in the extracted netlist:

```
PEX NETLIST SMASHED_DEVICE DELIMITER '@'
```

For a device M1 with four fingers in the layout, this would result in the following identifiers:

```
M1
M1@2
M1@3
M1@4
```

## PEX Netlist Subnode Section

Parasitic extraction

### PEX NETLIST SUBNODE SECTION {YES | NO}

#### Parameters

- **YES**

Required keyword that indicates nonstandard SPICE statements will be included in the subnode section of the netlist.

- **NO**

Required keyword that indicates nonstandard SPICE statements will not be included in the subnode section of the netlist.

#### Description

This statement specifies whether or not to include statements beginning with ‘\*|S’ in a DSPF or SPEF netlist. If PEX Netlist Subnode Section is not included in a rule file, then statements beginning with ‘\*|S’ appear in the DSPF or SPEF netlist in the subnode section.

#### Examples

This SVRF command will include netlist code with ‘\*|S’ at the beginning of the line:

```
PEX NETLIST SUBNODE SECTION YES
```

# PEX Netlist Unshort Device Pins

Parasitic extraction

**PEX NETLIST UNSHORT DEVICE PINS {*value* | NO}**

## Parameters

- ***value***

Required parameter that specifies the value of the nominal resistors connecting shorted device pins. The default value is 0.01 Ohm for CalibreView, DSPF, and SPEF formats. For all other netlist formats the value is not set. That is, no resistors are added to the netlist.

- **NO**

Required keyword that prevents the netlisting of resistors to unshort device pins. The default 0.01 Ohm resistors will not be used in the netlist. This behavior is limited to the CalibreView, DSPF, and SPEF formats.

## Description

Sets the value for nominal resistors connecting shorted device pins. A value greater than zero creates resistors with that value. Each device pin receives its own node name, and the small-valued resistor connects them.

## Examples

Include the following statement in your rule file to include 0.05 Ohm resistors for removing shorts between device pins:

```
PEX NETLIST UNSHORT DEVICE PINS 0.05
```

## PEX Netlist Uppercase Keywords

Parasitic extraction

**PEX NETLIST UPPERCASE KEYWORDS {NO | YES}**

### Parameters

- **NO**  
Specifies to leave element names and HSPICE and DSPF keywords in lower case text. This is the default.
- **YES**  
Specifies to output element names and HSPICE and DSPF keywords in upper case text.

### Description

Specifies when to upper case element names and HSPICE and DSPF keywords in an output netlist.

### Examples

Include the following statement in your rule file for uppercase keywords and element names:

```
PEX NETLIST UPPERCASE KEYWORDS YES
```

# PEX Netlist Virtual Connect

Parasitic extraction

## PEX NETLIST VIRTUAL CONNECT {NO | YES}

### Parameters

- **NO**  
Specifies leaving disjoint net model fragments unconnected. This is the default.
- **YES**  
Specifies connecting net model fragments created by Virtual Connect statements to the main net model using a resistor. The default resistor value is zero ohms. You can specify a different resistor value by using the [PEX Netlist Unshort Device Pins](#) SVRF statement.

### Description

Specifies how to handle disjoint net model fragments created by Virtual Connect statements. You should use this statement in limited cases, usually when testing subsets of a design for debugging extraction problems. Your final design must be LVS clean without using this statement.

#### **Caution**



The nodes used to connect the fragment to the net model are arbitrary. Do not use this statement during final sign-off simulations and verification.

For more information see the [Virtual Connect Name](#) statement and “`calibre -xrc -fmt`” in the [Calibre xRC User’s Manual](#) for invoking the `-fmt_warnings` option which lists connections made by the formatter.

### Examples

To attach floating pins while the root cause of connectivity is being identified, include the following statements in your rule file:

```
PEX NETLIST VIRTUAL CONNECT YES
PEX NETLIST UNSHORT DEVICE PINS 0.01
```

## PEX Pin Order

Parasitic extraction

**PEX PIN ORDER {LAYOUT | {**SOURCE** | **FILE** *filename* [**ALLPINS**]}}}**

### Parameters

- **LAYOUT**

Keyword that specifies the netlist pin ordering as layout. This is the default if this statement is not specified.

- **SOURCE**

Keyword that specifies the netlist pin ordering as source. Ideally, the number of pins for the cell in the source netlist should match the number of pins in the layout netlist.

- **FILE** *filename*

Keyword and parameter that specifies to use a given netlist file for ordering ports. The cell and port names in *filename* must match the names in the layout.

- **ALLPINS**

Optional keyword that allows all pin names from the source netlist to be included in the extracted netlist. This keyword cannot be combined with the LAYOUT keyword.

### Description

Used to control the pin ordering of the netlist. The default ordering is **LAYOUT**.

When using the **SOURCE** keyword, ideally, the number of source pins should match the number of layout pins. If the number of pins for the cell in the source netlist doesn't match the number of pins in the layout netlist, warnings are issued. The following conditions generate warnings:

- The number of source pins does not match the number of layout pins. If there are more layout pins than source pins, the extra layout pins are removed from the netlist.
- A layout port does not map to a source port. In this case the layout net is left floating.
- A source port does not map to a layout port. If there are more source pins than layout pins:
  - If the **ALLPINS** keyword was specified, the source port is netlisted and attached to a floating net.
  - If the **ALLPINS** keyword was not specified, the source port is omitted from the netlist.

This command cannot change pin names, only order.

### Examples

To match the source netlist used by LVS:

PEX PIN ORDER SOURCE

To read in a SPICE netlist used by the simulator:

```
PEX PIN ORDER FILE "$SIM_DIR/Eldo/fabric.spi"
```

To explicitly match the order of the layout:

PEX PIN ORDER LAYOUT

To include all pins from the source netlist:

```
PEX PIN ORDER SOURCE ALLPINS
```

To include all pins from a specified file:

```
PEX PIN ORDER FILE "$SIM_DIR/Eldo/fabric.spi" ALLPINS
```

## PEX Power

Parasitic extraction

**PEX POWER** [LAYOUT | SOURCE] *name* [*name* ...]

Used only in Calibre xL.

### Parameters

- LAYOUT

An optional keyword that specifies the net names are derived from the layout. This is the default if the keyword is not specified.

- SOURCE

An optional keyword that specifies the net names are derived from the schematic. Ensure the SVRF rule file contains a valid [Source Path](#) statement and a valid [Source Primary](#) statement identifying the primary cell when specifying this option.

- *name*

A required name of a power net. You can specify *name* any number of times in a statement, and the nets are independent of one another. The string *name* can contain one or more question mark (?) characters. This wildcard character matches zero or more characters. The *name* can be a string variable.

### Description

Specifies one or more power net names to consider during inductance extraction.

---

**Note**



If the PEX Power statement is not specified, the Calibre xL tool searches for [LVS Power Name](#) statements to identify power nets. If neither the PEX Power nor the LVS Power Name statements are specified, the Calibre xL tool identifies nets with the name VDD? and VCC? as power nets, where ? is a wildcard character.

---

### Examples

The following statement instructs the Calibre xL tool to use VDD and VCC as power nets.

```
PEX POWER VDD VCC
```

## PEX Probe File

Parasitic extraction

### **PEX PROBE FILE** *filename* [PROTECT]

Used only in Calibre xRC and Calibre xL.

#### Parameters

- *filename*

Required name of a file containing entries in the following format:

```
CELL cell_name
probe_name local_x local_y layername
CELL cell_name
[SINGULAR]
...
```

where:

- *cell\_name* is a valid extracted cell and a layout cell name.
- *probe\_name* must fall on a net for which parasitics are netlisted.
- a polygon exists on *layername*, at the specified coordinates (*local\_x*, *local\_y*), where the coordinates are given in user units.
- SINGULAR is an optional parameter that limits the probe to a single instance.

#### Note



During device extraction, LVS may promote polygons out of a cell. If you defined a probe point for a polygon, and the polygon got promoted out of the cell, the probe point you defined will not be connected. You can prevent this from happening by putting probe points at the highest level for which a net exists.

- PROTECT

An optional keyword that prevents probe points being removed when a probe point is at the same node as a pin or port. When PROTECT is specified, the Calibre xRC Formatter inserts a 0.01 ohm resistor between the probe point and the port or pin. (When PROTECT is not specified, the probe point is removed and a warning is printed.) The resistor value can be set using the [PEX Netlist Unshort Device Pins](#) statement.

#### Description

Used to verify timing from specific points on each net.

Within the parasitic model, each probe point exists as an internal connection with a unique *probe\_name*. You will be able to access each probe point by net and *probe\_name*, depending on your simulator's behavior.

## PEX Probe File

---

The Calibre xRC tool returns warnings when you are using PEX Probe File and the following conditions occur:

- If a duplicate probe point is found on a net, Calibre xRC ignores the duplicate.
- If a probe point is defined to be on a layer that was promoted out of a cell as a side effect of device extraction, Calibre xRC does not connect the probe point.
- If a probe point conflicts with a port, a device pin, or an hport and PROTECT is not set, Calibre xRC ignores the probe point.

When a probe point is defined inside a cell that is used in a hierarchical design, the probe is automatically replicated for all instances of the cell containing the probe. When the SINGULAR keyword is used, only a single probe appears in an arbitrary instance of the cell. The user does not choose which cell instance the probe appears in.

## Examples

```
PEX PROBE FILE testfile
```

## PEX Reduce Analog

Parasitic extraction

**PEX REDUCE ANALOG {NO | YES [DELAY\_ERROR *delay*] [NOISE\_ERROR *noise*]}**

### Parameters

- **NO**  
Keyword indicating to not apply this reduction. This is the default behavior for the Calibre xRC tool.
- **YES**  
Keyword indicating that analog reduction will be applied during the formatter step of parasitic extraction. This is the default behavior for the Calibre xACT 3D fieldsolver.
- **DELAY\_ERROR *delay***  
Optional keyword set specifying the timing delay threshold used for parasitic reduction. The *delay* units are seconds. The default is 0.5e-12 seconds (0.5 picoseconds).
- **NOISE\_ERROR *noise***  
Optional keyword set specifying the error threshold used for parasitic reduction. The noise error is defined as the amount of change in noise amplitude relative to a full strength signal. The value for *noise* is a ratio, a unitless floating number. The default is 0.01, or 1%.

### Description

Specifies performing reduction of extracted data from analog designs. This statement combines the effects of the [PEX Reduce TICER](#) and [PEX Reduce CC](#) SVRF statements. These latter statements take precedence over PEX Reduce Analog if they are all used in the same rule file. The [PEX Reduce Digital](#) statement is available for digital designs.

If PEX Reduce Analog and PEX Reduce Digital are in the rule file, Calibre xRC will output an error. If neither of these statements is used in a rule file, then no default reduction is done.

You may specify either or both parameters DELAY\_ERROR and NOISE\_ERROR. If you do not specify a value for either, the default is used.

Using [PEX Netlist](#) RLAYER, RLENGTH, RTHICKNESS, RLOCATION or RWIDTH with PEX Reduce Analog will generate a warning message and RLAYER, RLENGTH, RTHICKNESS, RLOCATION, or RWIDTH will be ignored.

For example, using PEX Netlist RWIDTH together with PEX Reduce Analog will output the following warning message and RWIDTH will be ignored:

```
RULES: PEX NETLIST "netlist.dist" DSPF 1e-6 LAYOUTNAMES GROUND VSS
LOCATION RWIDTH
RULES: PEX REDUCE ANALOG YES
WARNING: RWIDTH not supported with PEX REDUCE ANALOG
```

### Examples

#### Example 1

This statement specifies that analog RC reduction will not take place:

```
PEX REDUCE ANALOG NO
```

#### Example 2

This statement specifies that analog reduction will be done with the default values for DELAY\_ERROR (0.5 picosecond) and NOISE\_ERROR (1%):

```
PEX REDUCE ANALOG YES
```

#### Example 3

This statement specifies a reduction with a delay error of 5 picoseconds and a noise error of 15%:

```
PEX REDUCE ANALOG YES DELAY_ERROR 5e-12 NOISE_ERROR 0.15
```

#### Example 4

This example shows how one of the optional parameters may be used:

```
PEX REDUCE ANALOG YES DELAY_ERROR 5e-12
```

In this case the default value for NOISE\_ERROR (0.01) will be used.

## PEX Reduce CC

Parasitic extraction

### PEX REDUCE CC

{**ABSOLUTE** *threshold* | **RATIO** *percent* | **ABSOLUTE** *threshold* **RATIO** *percent*}  
[**SCALE** *scale*]

#### Parameters

- **ABSOLUTE threshold**

Keyword and threshold value. If the total coupling capacitance between a pair of nets is less than *threshold*, that coupling is applied as grounded capacitances to both nets. The *threshold* parameter is a positive floating point number scaled by the [Unit Capacitance](#) value. The default value for *threshold* is 3 femtofarads.

- **RATIO percent**

Keyword and threshold value as a decimal number between 0 and 1. The total coupled capacitance between two nets is compared to the total capacitance for each net individually. If for both nets the coupled capacitance accounts for less than *percent* (expressed as a decimal) of the total net capacitance, that coupling is applied as grounded capacitances to both nets.

- **SCALE scale**

Optional keyword and value that applies the *scale* factor to the reduced capacitances before grounding. Only the coupled capacitances that are converted to grounded capacitance are scaled; the other capacitances are not modified. By default, *scale* is 1.0.

#### Description

This optional reduction statement controls reduction of coupled capacitance between nets based on total capacitance. The reduction occurs during the formatting stage before the [PEX Reduce Digital TICER](#) reduction.

When both **ABSOLUTE** and **RATIO** are specified, both conditions must be true before the coupled capacitance is grounded. Both conditions must be in the same statement; it is an error to specify the statement once for each.

#### Examples

##### Example 1

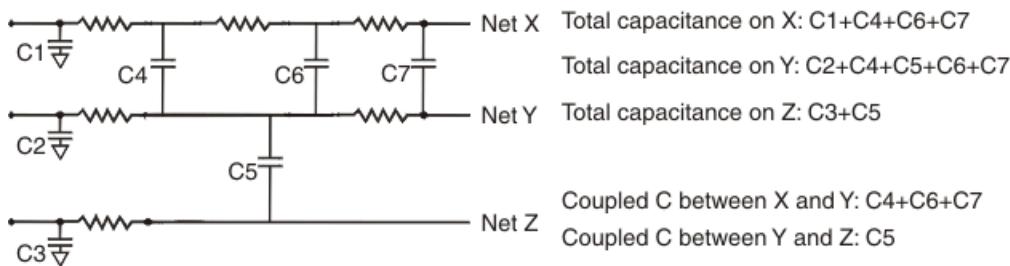
The **RATIO** condition is met when the total coupling capacitance between two nets is less than the specified *percent* on both. For example, net A and net B are coupled by 26 fF. The total capacitance (intrinsic and coupled) on net A is 250 fF and on net B is 300 fF.

```
PEX REDUCE CC RATIO 0.1      //Ground cc < 10% of net
```

The A-B coupling will not be grounded because although 26 fF is less than 10% of net B's 300 fF, it is greater than 10% of net A's 250 fF.

**Example 2**

The example nets in [Figure 4-295](#) illustrate coupling capacitance between nets and capacitance to ground (intrinsic capacitance). Based on geometry, nets may be broken into several RC sections. The total capacitance between two nets is the sum of all shared coupled capacitors.

**Figure 4-295. Coupling Capacitor Example**

When reducing the netlist using **ABSOLUTE threshold**, the coupled capacitance between nets is compared to the threshold value. For example, if the coupled capacitance between nets Y and Z  $C_5 < \text{threshold}$ , then after reduction nets Y and Z would have the following values:

$$\begin{aligned} CC_{YZ} &= 0 \\ C_Y &= (C_2 + C_5) + C_4 + C_6 + C_7 = \text{lumped capacitance} + CC_{XY} \\ C_Z &= (C_3 + C_5) \end{aligned}$$

where CC stands for total coupled capacitance and C for total capacitance. Because grounded capacitance is reported as a single value (lumped), the number of parasitics on the netlist is reduced in this case by 2 elements.

When reducing the netlist using **RATIO percent**, the ratio of coupled capacitance to total capacitance must be less than **percent** for both nets. For example, if **percent** is T then both parts of the following equation must be true before the coupled capacitance is grounded:

$$\frac{CC_{XY}}{C_X} < T \quad \text{AND} \quad \frac{CC_{XY}}{C_Y} < T$$

When both **ABSOLUTE threshold** and **RATIO percent** are specified, the coupled capacitance on the net must satisfy all three parts of the following condition:

$$\frac{CC_{XY}}{C_X} < T \quad \text{AND} \quad \frac{CC_{XY}}{C_Y} < T \quad \text{AND} \quad CC_{XY} < \text{threshold}$$

# PEX Reduce Digital

Parasitic extraction

## **PEX REDUCE DIGITAL NO**

**PEX REDUCE DIGITAL YES [DELAY\_ERROR *delay*] [NOISE\_ERROR *noise*]**

### Parameters

- **NO**  
Keyword indicating to not apply this reduction. This is the default for non-asic extraction.
- **YES**  
Keyword indicating that digital reduction will be applied during the formatter step of parasitic extraction. This is the default when the -asic switch is used.
- **DELAY\_ERROR *delay***  
Optional keyword set specifying the timing delay threshold used for parasitic reduction. The *delay* units are seconds. The default value is 1e-12 seconds (1 picosecond).
- **NOISE\_ERROR *noise***  
Optional keyword set specifying the error threshold used for parasitic reduction. The noise error is defined as the amount of change in noise amplitude relative to a full strength signal. The value for *noise* is a ratio, a unitless floating number. The default value is 0.03, or 3%.

### Description

Specifies performing a reduction of distributed RC extraction data so that there is no change in the time delay. This statement combines the effects of the [PEX Reduce TICER](#) and [PEX Reduce CC](#) SVRF statements. These latter statements take precedence over PEX Reduce Digital.

You may specify either or both parameters DELAY\_ERROR and NOISE\_ERROR. If you do not specify a value for either, the default is used.

PEX Reduce Digital is applied automatically when using LEF/DEF or Milkyway layout data is used unless the -noasic switch is specified during the parasitic extraction step (-pdb). If your rule file contains PEX Reduce Digital but you are not using the ASIC flow then the statement is applied during the formatter stage and the netlist is reduced.

Using [PEX Netlist RLAYER](#), RLENGTH, RTHICKNESS, RLOCATION or RWIDTH with PEX Reduce Digital will generate a warning message and RLAYER, RLENGTH, RTHICKNESS, RLOCATION, or RWIDTH will be ignored.

For example, using PEX Netlist RLAYER together with PEX Reduce Digital will output the following warning message and RLAYER will be ignored:

```
RULES: PEX NETLIST "netlist.dist" DSPF 1e-6 LAYOUTNAMES GROUND VSS
LOCATION RLAYER
RULES: PEX REDUCE DIGITAL YES
WARNING: RLAYER not supported with PEX REDUCE DIGITAL
```

## **Examples**

### **Example 1**

This statement specifies that digital RC reduction will not take place:

```
PEX REDUCE DIGITAL NO
```

### **Example 2**

This statement specifies that digital reduction will be done with the default values for DELAY\_ERROR (1 picosecond) and NOISE\_ERROR (5%):

```
PEX REDUCE DIGITAL YES
```

### **Example 3**

This statement specifies a reduction with a delay error of 5 picoseconds and a noise error of 15%:

```
PEX REDUCE DIGITAL YES DELAY_ERROR 5e-12 NOISE_ERROR 0.15
```

### **Example 4**

This example shows how one of the optional parameters may be used:

```
PEX REDUCE DIGITAL YES DELAY_ERROR 5e-12
```

In this case the default value for NOISE\_ERROR (0.03) will be used.

## PEX Reduce Distributed

Parasitic extraction

**PEX REDUCE DISTRIBUTED TICER** *frequency* [PORTMERGE [*range*]]

---

**Note**

 PEX Reduce Distributed has been renamed to [PEX Reduce TICER](#). There are no changes to parameter definitions and functionality. PEX Reduce Distributed has been deprecated as of the 2008.3 release.

---

## PEX Reduce Mincap

Parasitic extraction

### PEX REDUCE MINCAP {COMBINE | REMOVE} *threshold*

Used only in Calibre xRC.

#### Parameters

- **COMBINE**

Keyword that causes capacitors with value below *threshold* to be added together and reported as a single parasitic capacitor. See the description.

- **REMOVE**

Keyword that causes capacitors with value below *threshold* to be removed from a net. Intrinsic and coupled capacitors are handled differently as described below.

- ***threshold***

A required non-negative, floating point value that specifies a capacitor threshold. The value is in user-defined capacitance units specified by the [Unit Capacitance](#) statement. If you do not specify Unit Capacitance, then picofarads is used.

#### Description

Combines or removes extracted capacitors based on a user-defined threshold value. This command can be used to reduce the size of the netlist by reducing the number of extracted capacitors. Combining and removing capacitors take place during the formatting step of extraction.

This reduction technique cannot be used when the netlist includes parasitic self or mutual inductance as extracted by Calibre xL.

#### Combining Capacitors

When you specify a threshold value for PEX Reduce Mincap COMBINE, the extracted intrinsic capacitors that fall below this value are combined with the intrinsic capacitor of the nearest available node with an intrinsic capacitor.

For a net pair with extracted coupled capacitance, coupled capacitors less than the threshold value will be added to the nearest coupled capacitor on the same net pair.

For both intrinsic and coupled capacitors, if there is only a single combined capacitor left and the value falls below the threshold, then the capacitor is kept as is.

#### Removing Capacitors

The results of using PEX Reduce Mincap REMOVE are the same for intrinsic and coupled capacitors.

If the value of a capacitor is less than the user defined threshold, the capacitor will not be included in the netlist during formatting.

## SVRF Statement Precedence

The order reduction occurs in is as follows:

- If both PEX Reduce Coupled and PEX Reduce Mincap are in the rule file, PEX Reduce Coupled is ignored.
- PEX Reduce TICER runs.
- PEX Reduce Mincap COMBINE runs.
- PEX Reduce Mincap REMOVE runs.

## Examples

Assuming Unit Capacitance of picofarads, to combine extracted capacitors falling below 10 fF, use:

```
PEX REDUCE MINCAP COMBINE 0.01
```

Likewise, to remove capacitors on nets that fall below 20 fF, use:

```
PEX REDUCE MINCAP REMOVE 0.02
```

## PEX Reduce Minres

Parasitic extraction

### PEX REDUCE MINRES {COMBINE | SHORT} *threshold*

Used only in Calibre xRC.

#### Parameters

- **COMBINE**

Causes resistors with values below *threshold* to be added together and reported as a single parasitic resistor.

- **SHORT**

Causes resistors with values below *threshold* to be shorted.

- ***threshold***

A non-negative, floating point value that specifies a resistor threshold for COMBINE or SHORT. The [Unit Resistance](#) SVRF statement sets the units for this value.

#### Description

Combines or shorts extracted resistors based on a user-defined threshold value, preserving the parasitic resistance along the net. Use this statement to reduce the size of the netlist by reducing the number of parasitic resistors. The reduction takes place during the formatting step. For both options, this statement is applied before [PEX Reduce TICER](#) reduction.

This reduction method cannot be used when the netlist includes parasitic self or mutual inductances that are extracted by the Calibre xL tool.

Using this statement will not remove all resistors smaller than the specified threshold in all cases. For example, it will not remove small resistors joining two pins, a pin to a junction, or two junctions. A junction is defined as a node with more than two incident resistors.



#### Note

PEX Reduce Minres replaces the deprecated [PEX Tolerance Distributed R](#) statement. If both statements are used, PEX Reduce Minres will override PEX Tolerance Distributed R.

---

PEX Reduce Minres reduces resistive networks so they are electrically equivalent and topologically unchanged. This simpler reduction method differs from the method used by [PEX Reduce Digital](#) in that it reduces the network without complex analysis or error control. Also, because of the simple way in which it handles capacitors, it does not necessarily preserve interconnect delay.

Protected nodes are not reduced. A node is protected if it connects more than two resistors or contains an hprobe or a device pin.

This reduction method combines all unprotected resistors of resistance less than **threshold** into a neighboring larger resistor. Associated capacitors are split equally between two neighboring resistors.

Using [PEX Netlist](#) RLAYER, RLENGTH, RTHICKNESS, RLOCATION or RWIDTH with PEX Reduce Minres COMBINE will generate a warning message and RLAYER, RLENGTH, RTHICKNESS, RLOCATION, or RWIDTH will be ignored.

For example, using PEX Netlist RLOCATION together with PEX Reduce Minres COMBINE will output the following warning message and RLOCATION will be ignored:

```
RULES: PEX NETLIST "netlist.dist" DSPF 1e-6 LAYOUTNAMES GROUND VSS
LOCATION RLOCATION
RULES: PEX REDUCE MINRES COMBINE 10
WARNING: For asic mode, turning on PEX REDUCE DIGITAL by default
WARNING: RLOCATION not supported with PEX REDUCE DIGITAL
WARNING: RLOCATION not supported with PEX REDUCE MINRES COMBINE
```

## Examples

### Example 1

To combine extracted resistors falling below 0.5 ohms, include the following in your rule file:

```
PEX REDUCE MINRES COMBINE 0.5
```

### Example 2

To short any resistor with resistance less than 0.6 ohms, include the following in your rule file:

```
PEX REDUCE MINRES SHORT 0.6
```

## PEX Reduce ROnly

Parasitic extraction

### PEX REDUCE RONLY [SPARSIFY]

#### Parameters

- SPARSIFY

Optional keyword directing that the parasitics should be further reduced by combining resistors.

#### Description

Reduces parasitic R-only data by eliminating all internal nodes.

The PEX Reduce ROnly rule operates only on resistive networks, ones that do not include capacitance or inductance. It is most effective on networks with few ports and many internal nodes.

The ROnly operation reduces parasitic netlists so that they are electrically equivalent to the originals, but the topology may be entirely different. The effective resistance between ports is preserved.

If [PEX Extract Temperature](#) and [PEX Netlist...TCOEFF](#) are both also present in the SVRF rule file, PEX Reduce ROnly performs temperature-based reduction as described in PEX Extract Temperature. The temperature coefficients are calculated for each reduced element, and the parasitic resistors contain the resistance value at the nominal temperature. The netlists include TC1 and TC2 per resistor, not modifying the resistance value.

With SPARSIFY, the netlist is further reduced by eliminating some resistors and adjusting the values of others to compensate.

Layer information is lost, even if RLAYER is selected with PEX Netlist. Using [PEX Netlist RLAYER](#), [RLENGTH](#), [RTHICKNESS](#), [RLOCATION](#) or [RWIDTH](#) with PEX Reduce ROnly will generate a warning message and RLAYER, RLENGTH, RTHICKNESS, RLOCATION, or RWIDTH will be ignored.

For example, using PEX Netlist RLAYER together with PEX Reduce ROnly will output the following warning message and RLAYER will be ignored:

```
RULES: PEX NETLIST "netlist.dist" DSPF 1e-6 LAYOUTNAMES GROUND VSS
LOCATION RLAYER
RULES: PEX REDUCE RONLY
WARNING: For asic mode, turning on PEX REDUCE DIGITAL by default
WARNING: RLAYER not supported with PEX REDUCE RONLY
```

The ROnly reduction is more aggressive than [PEX Reduce Digital](#), making it especially useful for large networks, such as power distribution networks, that might otherwise exceed available computer resources. When both reductions occur in the SVRF rule file, PEX Reduce ROnly is applied to pure resistive networks, and [PEX Reduce TICER](#) to the other networks.

## **Examples**

PEX REDUCE RONLY

## PEX Reduce TICER

Parasitic extraction

**PEX REDUCE TICER** *frequency* [PORTMERGE [*range*]]

### Summary

Specifies to perform reduction of distributed RC extraction data in a way that has little impact on circuit characteristics such as delay up to a specified frequency.

#### Note



[PEX Reduce Digital](#) is the preferred method for performing digital reduction. It combines the effects of both PEX Reduce TICER and [PEX Reduce CC](#) SVRF statements.

---

### Parameters

- *frequency*

The required *frequency* parameter is a user-defined calculated number in Hertz, expressed using scientific notation. For example:

```
PEX REDUCE TICER 40e9
```

The *frequency* parameter controls which nodes in the circuit the tool can select for subsequent elimination; specifically, the tool selects nodes with time constants less than the *frequency* parameter.

- Setting the *frequency* parameter to a higher value results in larger (more R and C elements) interconnect circuits having a wider bandwidth of accuracy.
- Setting the *frequency* parameter to a lower value results in more compression and an earlier roll-off in accuracy.

Calculate the *frequency* parameter using the following formula:

$$\textit{frequency} = \frac{\textit{tradeoff\_value}}{\textit{transition\_time\_minimum}}$$

where

*tradeoff\_value* — a number between 4 and 10. A larger *tradeoff\_value* results in a higher *frequency* parameter value and, consequently, more accurate and less aggressive reduction.

*transition\_time\_minimum* — the shortest rise or fall time expected in the design. In general, you can estimate this value using 1/5 of the design's switching delay.

TICER reduction preserves the frequency response of an RC interconnect circuit from dc up to *frequency*. Consequently, the *frequency* parameter controls trading off compression with accuracy.

- PORTMERGE [*range*]

An optional keyword that enables a more aggressive form of TICER reduction. This form combines ports whenever the change in timing is less than *range*. The default *range* (timing tolerance) is such that the impact on the time constants of ports is equivalent to that set by *frequency* for nodes.

The *range* parameter is specified in a derived time unit equal to R units times C units. For example, if R is in ohms and C is in picofarads, the derived time unit is picoseconds. The default value of range uses the following formula:

$$\text{range} = \frac{1}{8} \langle \text{frequency} \rangle^{-1}$$

For example, if *frequency* is 25 GHz ( $25 \times 10^9$  cycles/second), the inverse is  $4 \times 10^{-11}$  seconds/cycle and the default range is 5 picoseconds, or one-eighth of this value.

## Description

Specifies to perform parasitic element reduction of distributed RC extraction data such that there is very little change in the time delay up to the specified frequency.

This specification statement activates electrically-based reduction. The reduction compresses nets into electrically equivalent ones although the topology may be entirely different. That is, the reduced netlist may not map pin-for-pin or element-for-element to the original netlist. Reduction does not preserve exact resistance values, RLOCATION, RWIDTH, or RLAYER.

Basic TICER does not move or alter ports. Port merging enables TICER to achieve more reduction than it might otherwise by combining certain ports that normally would be kept separate. In practice, many circuits—especially local networks—are “port bound” in the sense that they have relatively many ports and relatively few internal nodes. Often basic TICER cannot reduce these nets much because it can only eliminate internal nodes. With port merging, TICER can eliminate ports as well by transferring the ports onto neighboring nodes. This gives TICER reduction greater flexibility and allows it to achieve more reduction.

If [PEX Extract Temperature](#) and [PEX Netlist...TCOEFF](#) are both also present in the SVRF rule file, PEX Reduce TICER performs temperature-based reduction as described in [PEX Extract Temperature](#). The temperature coefficients are calculated and reported for each reduced element, and the parasitic resistors are given as nominal values (that is, the values at the nominal temperature). The netlists include TC1 and TC2 for each resistor that depends on temperature.

### Note



When using TICER to reduce a netlist which includes temperature sensitivity, the final netlist will contain extracted capacitors with slightly changed values due to temperature coefficients. This is a consequence of the reduction process. For more details, see “[TICER and Temperature Sensitivity Effects](#)” in the [Calibre xRC User’s Manual](#).

Using [PEX Netlist](#) RLAYER, RLENGTH, RTHICKNESS, RLOCATION or RWIDTH with PEX Reduce TICER will generate a warning message and RLAYER, RLENGTH, RTHICKNESS, RLOCATION, or RWIDTH will be ignored.

## PEX Reduce TICER

For example, using PEX Netlist RTHICKNESS together with PEX Reduce TICER will output the following warning message and RTHICKNESS will be ignored:

```
RULES: PEX NETLIST "netlist.dist" DSPF 1e-6 LAYOUTNAMES GROUND VSS  
LOCATION RTHICKNESS  
RULES: PEX REDUCE TICER 1e-12  
WARNING: For asic mode, turning on PEX REDUCE DIGITAL by default  
WARNING: RTHICKNESS not supported with PEX REDUCE DIGITAL  
WARNING: RTHICKNESS not supported with PEX REDUCE TICER
```

In Calibre xRC, reduction occurs during netlist generation. The resulting netlists contain smaller parasitic models than they would otherwise.

The primary advantages of reduction are:

- Makes the reports easier to read.
- Results in smaller netlists, thus faster simulations.
- Requires less memory resources (uses less RAM and reduces swap space allocations).

### Note

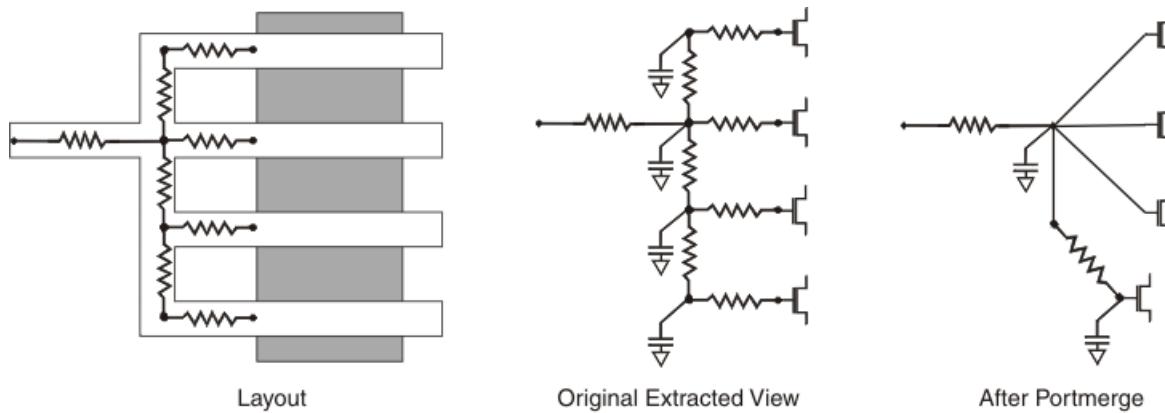


It is especially important when using PORTMERGE to choose a correct setting for the TICER frequency. If there are too many low-value resistors in the netlist, omit PORTMERGE from the PEX Reduce TICER statement or use a higher TICER frequency, for example, 1e25.

## Examples

The networks in [Figure 4-296](#) show the change in a multi-fingered transistor due to port merging. Because multifingered devices usually have roughly as many ports as internal nodes, basic TICER cannot provide much reduction. Port merging connects multiple ports to the same node.

**Figure 4-296. Example Circuit Reduced with TICER PORTMERGE**



As shown in [Figure 4-296](#), the multifingered transistor's parasitic network, representing only the interconnect, is 9 nodes separated by parasitic resistors. Where the fingers pass into individual transistors, there are ports connecting to device models of a transistor. The internal nodes are also the connection point for the intrinsic parasitic capacitance.

After reduction with the PORTMERGE option, the internal nodes on the top three fingers have been merged but the bottom finger remains separate because the change on timing was greater than *range*. Capacitance on the remaining nodes is greater, including the capacitance of the merged nodes.

# PEX Reduce Via Resistance

Parasitic extraction

**PEX REDUCE VIA RESISTANCE** [*layer1 layer2*]  
  {**STANDARD** [*COUNT max*] [*DISTANCE value*] |  
  **FLEXIBLE** [*DISTANCE value*] [*MINSTEP value*] |  
  **MINIMUM** |  
  **OFF**}

## Parameters

- *layer1 layer2*

An optional keyword pair specifying a particular set of vias to which the statement applies. The layers must be connected to the vias and have connectivity and resistance. If *layer1* *layer2* are not present, the rule applies to all vias in the design.

- **STANDARD** [*COUNT max*] [*DISTANCE value*]

A required keyword that specifies standard via reduction. A cluster of vias is reduced to one via.

The optional keyword set *COUNT max* limits the number of vias that are reduced to a single via. The default is no limit.

The optional keyword set *DISTANCE value* specifies the minimum distance in microns that defines clusters of vias. The default is 5 microns.

- **FLEXIBLE** [*DISTANCE value*] [*MINSTEP value*]

A required keyword that specifies flexible via reduction. FLEXIBLE reduction takes the original via cluster, breaks it up into rectangular arrays, and then reduces all of the vias to four vias, located on the top, left, bottom, and right of each rectangular section. FLEXIBLE reduction is equivalent to FLEXIBLE MINSTEP 0.

The optional keyword set *DISTANCE value* specifies the minimum distance in microns that defines clusters of vias. The default is 5 microns.

The optional keyword set *MINSTEP value* breaks the via array into square arrays (*n x n*), where *n* is the original width of the via array. It breaks it into as many squares as possible. Any remaining shapes are rectangular. Within each of these regions, all vias are reduced to one via that is geometrically as close to the center of the region as possible. The *value* specifies the minimum number of vias that are reduced to a single via. The default is 3.

If PEX Reduce Via Resistance statement is not specified in the rule file, then PEX Reduce Via Resistance FLEXIBLE MINSTEP 3 is enabled by default.

- **MINIMUM**

A required keyword that minimizes via reduction. In most cases, vias are modeled individually. However, if several vias in one column are perpendicular to the metal path axis, the resistances for these vias will be shorted to the same node, effectively connecting the vias in parallel.

- **OFF**

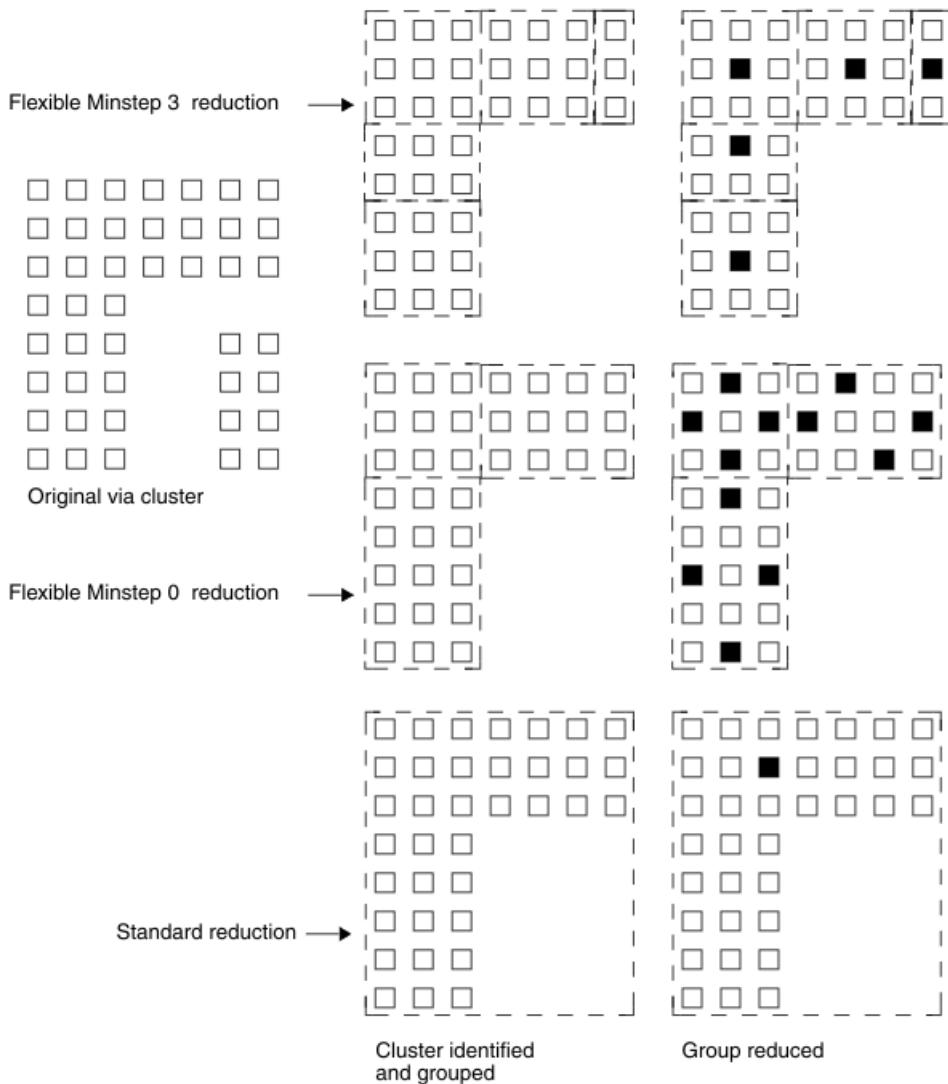
A required keyword which turns off via reduction. Turning via reduction off will create a netlist that contains resistors representing all vias in the layout. Using this keyword may cause a substantial increase in parasitic database and netlist size. To prevent diffusion resistors from appearing in the output netlist, set [PEX Netlist Unshort Device Pins NO](#). It is recommended to use this keyword when specifying layers. This keyword is not recommended for use on designs with large via clusters.

## Description

Specifies how to handle clusters of identically-connected vias when calculating resistance. (It does not affect capacitance extraction.) Use this operation without layers to indicate a default reduction for all vias, and also override it for particular via layers with additional statements indicating connecting layer pairs.

During PDB generation, the extraction software clusters adjacent vias, potentially subdivides the cluster into groups, and reduces each group to a single resistor in standard via reduction or four edge-located resistors in flexible via reduction as shown in [Figure 4-297](#).

The default is to reduce any cluster to one node, in the approximate center. A cluster is defined by default as a group of vias with a gap of at least 5 microns to the next group of vias. (The cluster itself may extend considerably beyond 5 microns.)

**Figure 4-297. Flexible Versus Standard Via Reduction**

The optional DISTANCE *value* keyword set identifies clusters; vias that are within the limit distance are considered to be part of the cluster, and vias further away are not. For example, if *value* had been set too large in Figure 4-297, the 2x4 via block would have been treated as part of the larger cluster. For most applications, a typical reduction limit is five times the minimum via spacing.

For **FLEXIBLE** via reduction, groups are determined by geometry. As illustrated in Figure 4-297, each group is a rectangle, square, or line. Vias that could be included in more than one rectangle form their own group. Each group is then reduced to four vias, which are placed around the outer edge of the group. (For lines, only two points are used.) The optional keyword set MINSTEP *value* controls how many vias in the cluster are kept. A *value* of 1 is the most conservative and generates a larger netlist with more vias. A larger *value* is less conservative and generates a smaller netlist as the number of vias are reduced.

For **STANDARD** via reduction, grouping and reduction is controlled with the optional keyword set COUNT *max*. The location of the via halfway through *max* is used for the reduced vias. This location may be at an edge of the overall cluster or in the middle.

## Examples

### Example 1

To make the default behavior explicit in your rule file, add the following line:

```
PEX REDUCE VIA RESISTANCE STANDARD DISTANCE 5
```

### Example 2

To use flexible via reduction by default, but minimize via reduction for contacts in order to analyze sensitive connections, the rule file might contain the following lines:

```
//Derive layers using seed shapes on layer "crit_area". Connectivity and  
//resistance inherited from m1 and poly respectively.  
top_preserve = m1 AND crit_area  
bot_preserve = poly AND crit_area  
PEX REDUCE VIA RESISTANCE FLEXIBLE  
PEX REDUCE VIA RESISTANCE top_preserve bot_preserve MINIMUM
```

### Example 3

To turn off via reduction for all layers add the following line to your rule file:

```
PEX REDUCE VIA RESISTANCE OFF
```

### Example 4

To turn off via reduction between layers metal1 and metal2, add the following line to your rule file:

```
PEX REDUCE VIA RESISTANCE metal1 metal2 OFF
```

# PEX Report

Parasitic extraction

**PEX REPORT** *filename* {LAYOUTNAMES | SOURCENAMES}

## Parameters

- *filename*

A required filename for the report.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

If the *filename* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

- **LAYOUTNAMES**

Required argument that specifies the names are derived from the layout.

- **SOURCENAMES**

Required argument that specifies the names are derived from the source. When you specify the **SOURCENAMES** option, a [Source Primary](#) statement must identify a valid primary cell in the rule file.

## Description

Specifies generating a report for parasitic results of the extracted circuit and writing the report in the specified file.

## Examples

To report extracted parasitics based on source names, include the following statements in your rule file:

```
SOURCE PRIMARY "myDesign"  
PEX REPORT myDesign.report SOURCENAMES
```

# PEX Report Coupling Capacitance

Parasitic extraction

## PEX REPORT COUPLING CAPACITANCE *filename*

[NUMBER *count*]  
 [SPLIT\_NET]  
 [THRESHOLD *value*]  
 [LAYOUT | SOURCE]

### Parameters

- *filename*

A required output file name for the report.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

- NUMBER *count*

An optional keyword and parameter that specify the maximum number of coupled capacitors to include in the report. The default value for *count* is 1000.

- SPLIT\_NET

An optional keyword that controls the output format of the report. When SPLIT\_NET is used, the ratio of coupling capacitance for Net B is shown on a separate line.

- THRESHOLD *value*

An optional keyword and parameter that specifies the threshold for coupling capacitance shown in the report. Capacitance values below this threshold are not shown.

- LAYOUT

An optional keyword that specifies using net names from the layout in the report. This is the default behavior for non-ASIC extraction.

- SOURCE

An optional keyword that specifies using net names from the source netlist in the report. For ASIC extraction, this is the default behavior.

### Description

This statement generates a report of the ratio of coupling capacitors and total net capacitance between pairs of nets in a layout. Net names may come from the layout or the source netlist.

The report shows the ratio of coupling capacitance between any two nets related to the total net capacitance for each net. For nets A and B, the ratios are:

$$\text{Percentage A} = [\text{CC}_{AB} / C_{A\_total}] \times 100$$

$$\text{Percentage B} = [\text{CC}_{AB} / C_{B\_total}] \times 100$$

where:

- *Percentage A* is the amount of coupling capacitance between nets A and B based on the total intrinsic and coupled capacitances on net A.
- *Percentage B* is the amount of coupling capacitance between nets A and B based on the total intrinsic and coupled capacitances on net B.
- $CC_{AB}$  is the coupling capacitance between nets A and B.
- $C_{A\_total}$  is the total parasitic capacitance, coupling plus intrinsic, on net A.
- $C_{B\_total}$  is the total parasitic capacitance, coupling plus intrinsic, on net B.

## Examples

### Example 1

In this example, the report includes net-to-net coupling capacitors between pairs of nets in the circuit:

```
PEX REPORT COUPLING CAPACITANCE "coupledCaps.rep"
```

The output generated from this statement is:

Ratio from A (%)	Ratio from B (%)	Coupling Cap (F)	NetA	NetB
14.1766	3.19351	2.627139e-15	7	VSS
8.58376	2.21703	1.823839e-15	8	VSS
6.81333	7.81192	1.447666e-15	8	7

Line one of the report shows about 2.6 femtofarads of coupled capacitance between net 7 and VSS. The contribution of this capacitance to the total parasitic capacitance on 7 is about 14%. Line one of the report also shows that the reported coupling capacitance between 7 and VSS accounts for about 3.2% of the total capacitance on VSS.

### Example 2

In this example, the report includes coupling capacitors on Net A that have a value greater than 1.5 femtofarads:

```
PEX REPORT COUPLING CAPACITANCE "splitNets.rep" SPLIT_NET  
THRESHOLD 1.5e-15
```

The output generated from this statement is:

Ratio from A (%)	Coupling Cap (F)	NetA	NetB
14.1766	2.627139e-15	7	VSS
8.58376	1.823839e-15	8	VSS
3.19351	2.627139e-15	VSS	7
2.21703	1.823839e-15	VSS	8

The ratios of coupled capacitance reported between net 7 and VSS are split into two separate lines in the report shown on line one and line three.

# PEX Report Distributed

Parasitic extraction

**Note**



As of the 2009.1 release, this statement has been deprecated. Use [PEX Report](#) instead.

**PEX REPORT DISTRIBUTED** *filename* ASCII {ID | LAYOUT | SOURCE}

**PEX REPORT DISTRIBUTED NONE**

## Parameters

- ***filename***

A required filename for the report.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

If the ***filename*** ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

- **ASCII**

Keyword that specifies the report format as ASCII and lists the distributed extraction results

- **ID**

Required argument that specifies the names are derived from layout handles. May not be specified with either **LAYOUT** or **SOURCE**

- **LAYOUT**

Required argument that specifies the names are derived from the layout. May not be specified with either **ID** or **SOURCE**.

- **SOURCE**

Required argument that specifies the names are derived from the schematic. When specifying the **SOURCE** option, make sure that a valid [Source Primary](#) statement identifying a valid primary cell is in the rule file.

- **NONE**

A required keyword if ***filename*** is not specified. Indicates no distributed report is generated. This is the default behavior if the statement is not specified.

## Description

Specifies generating a report for distributed RC parasitic results of the extracted circuit and writing the report in the specified file.

### Note



When you use this statement with Calibre xRC, the generated reports will *exclusively* contain capacitance information.

---

**ID** and **LAYOUT** are usually identical, but the separate keywords are preserved for backwards compatibility with older rule files.

If you are performing a lumped C extraction run, then the tool ignores this statement.

This statement can cause longer runtimes for large designs.

## Examples

```
PEX REPORT DISTRIBUTED report.dist ASCII LAYOUT
```

# PEX Report Lumped

Parasitic extraction

**Note**



As of the 2009.1 release, this statement has been deprecated. Use [PEX Report](#) instead.

## PEX REPORT LUMPED {NONE | *filename* {ID | LAYOUT}}

### Parameters

- ***filename***

A required filename for the report.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

If the ***filename*** ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

- **ID**

Keyword that specifies the names are derived from layout handles.

- **LAYOUT**

Keyword that specifies the names are derived from the layout.

- **NONE**

A required keyword if ***filename*** is not specified. Indicates no report is generated. This is the default behavior if the statement is not specified.

### Description

Specifies generating a report for lumped C parasitic results of the extracted circuit and writing the report in the specified file.

If you are performing a distributed RC extraction run, then the tool ignores this statement.

**ID** and **LAYOUT** are usually identical, but the separate keywords are preserved for backwards compatibility with older rule files.

### Examples

```
PEX REPORT LUMPED "report.lump" LAYOUT
```

# PEX Report Mutual Inductance

Parasitic extraction

**PEX REPORT MUTUAL INDUCTANCE** *filename* [LAYOUTNAMES |  
SOURCENAMES]

Used only in Calibre xL.

## Parameters

- *filename*

A required filename for the report.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

If the *filename* ends in either the .Z or .gz suffix, then the file is automatically compressed using the UNIX compress or gzip commands, respectively. This assumes these commands are available in your \$PATH.

- LAYOUTNAMES

An optional keyword that specifies using net names from the layout in the report. This is the default behavior.

- SOURCENAMES

An optional keyword that specifies using net names from the source netlist in the report.

## Description

Triggers the generation of a report containing information about mutual inductance on a per-net basis and writing the report in the specified file. This ASCII text file will only list the victims as specified in the victim file with [PEX Inductance Victim](#), or [PEX Inductance Victim Path](#) statements. The self inductance, total mutual inductance, and mutual inductance due to each aggressor will be listed for each net. The entries in the report are sorted in descending order of total mutual inductance and are tab delimited so the report may be imported into a spreadsheet.

## Example

To report extracted mutual inductance on a per-net basis, include the following statement in your rule file:

```
PEX REPORT MUTUAL INDUCTANCE myMutual.report
```

The output generated from this statement is:

VICTIM	AGGRESSOR	M	SELF_L	TOTAL_M
LL			841	29.45163
	RR	17.0681		
	VTAIL	7.60326		
	L_D	2.75629		
	L_G	2.02398		

---

RR		1431	24.04532
LL	17.0681		
VTAIL	3.67341		
L_D	1.99198		
R_G	1.31183		

The VICTIM column contains the victim net name. The AGGRESSOR column lists the aggressor net. The M column is the total mutual inductance between the victim and the aggressor nets. The SELF\_L column contains the total self inductance of the net and the TOTAL\_M column contains the total mutual inductance of the victim to all other nets.

# PEX Report Netsummary

Parasitic extraction

## PEX REPORT NETSUMMARY *filename*

[FULL | LOCAL | ALL]

[DETAIL | SUMMARY]

[{LAYOUT | SOURCE} *net* [*net...*] | NETFILE *nets\_file*]

[CELL *cellname*]

[SCALE *value*]

[COLUMNS {BASIC | ADVANCED}]

## Summary

Generates a report for the parasitic capacitance for the specified cells and nets. This statement can be specified multiple times.

## Parameters

- ***filename***

A required filename for the report. The PEX Report Netsummary statement does not support automatic file compression.

The ***filename*** parameter can contain environment variables. For information regarding the use of environment variables in the ***filename*** parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

- **FULL | LOCAL | ALL**

An optional keyword that specifies the scope of the report. Only one of the following may be specified:

FULL — The report contains the parasitic capacitance of the specified nets within *cellname* and its cells. This is the default behavior if the option is not specified.

LOCAL — The report contains the parasitic capacitance of the specified nets for *cellname* only. If a net crosses into a subcell, those parasitics are not included.

ALL — The report contains the parasitic capacitance of the specified nets within *cellname*, and the specified nets and any additional nets within *cellname*’s cells.

- **DETAIL | SUMMARY**

An optional keyword that specifies whether the nets’ capacitances are itemized by sections or only reported for the whole net. The default is DETAIL, which lists values for each of the sections of each net. The total capacitance and total coupled capacitance values for the topmost section of the net include the capacitances of child sections.

- **{LAYOUT | SOURCE} *net***

An optional keyword and parameter specifying particular nets to report. LAYOUT specifies the net names are based on the layout. SOURCE specifies the net names are based on the source netlist. You can specify *net* one or more times separated by spaces. Wildcard characters are not supported. Only the specified nets appear in the summary.

The default behavior is to report all nets. This keyword set cannot be specified with the NETFILE keyword set.

- **NETFILE** *nets\_file*

An optional keyword and parameter that specifies a file containing net names. Only those nets are reported in the summary. Each net name should be on a separate line in the file, and preceded by LAYOUT or SOURCE. Cannot be specified with LAYOUT or SOURCE *net*.

The *nets\_file* parameter can contain environment variables. For information regarding the use of environment variables in the *nets\_file* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

- **CELL** *cellname*

An optional keyword and parameter that specifies a cell. Only nets within that cell will be reported. Wildcard characters are not supported. The default value is the cell specified in the [Layout Primary](#) statement, which must not use the wildcard character “\*”.

- **SCALE** *value*

An optional keyword and floating point number that is multiplied with the capacitive values reported. The default value is 1.0.

- **COLUMNS** {BASIC | ADVANCED}

An optional pair of keywords which control the report output.

**BASIC** — Report total capacitance, total coupled capacitance, and the ratio of coupled capacitance to total capacitance only. The total capacitance includes both coupled capacitance and intrinsic capacitance.

**ADVANCED** — Report total capacitance, total coupled capacitance, the ratio of coupled capacitance to total capacitance, capacitance for the portion of *net* within *cellname*, coupled capacitance for the portion of *net* within *cellname*, capacitance for the portion of *net* within *cellname*'s cells, and coupled capacitance for the portion of *net* within *cellname*'s cells.

## Description

Specifies to generate a report which details the parasitic capacitance values as stored in the parasitics database (PDB). The parasitic capacitance values reflect the reductions specified in the rule file. By default, all nets are reported. If the extraction was done in full hierarchical mode, the parasitic capacitance is further detailed by cell. If a resistance-only extraction was done, the report is still generated.

## Examples

### Example 1

To output a report on the capacitances of all extracted nets:

```
MASK SVDB DIRECTORY svdb QUERY XRC SLPH  
PEX REPORT NETSUMMARY net.summary
```

**Example 2**

To output a report on the power nets:

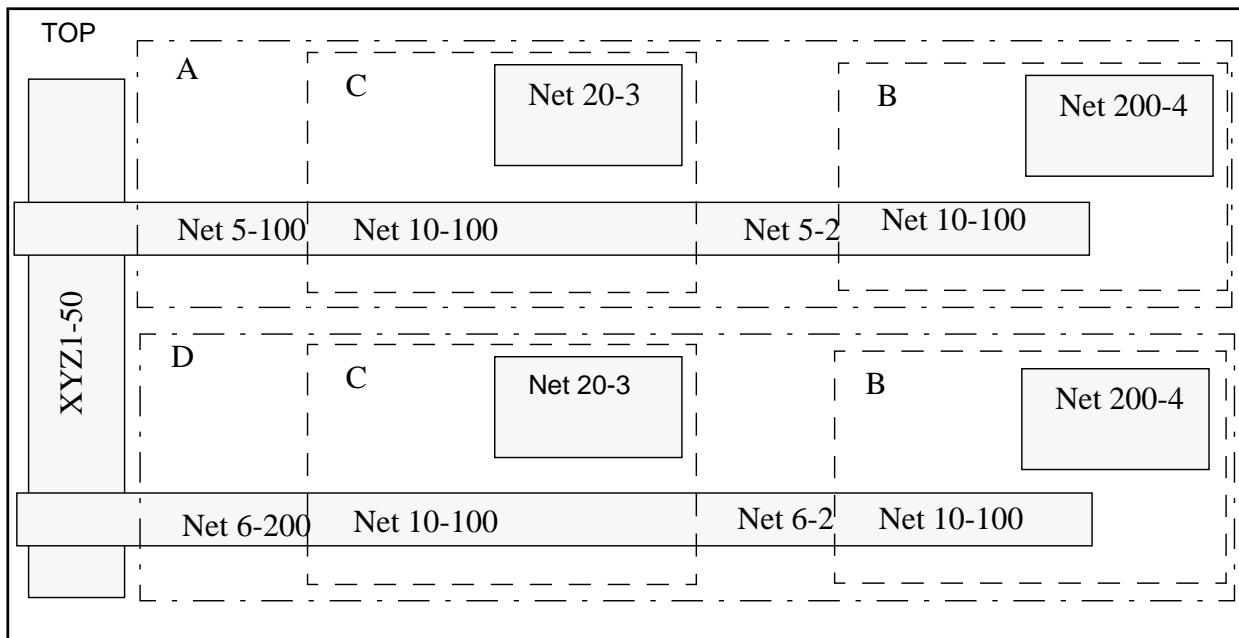
```
PEX REPORT NETSUMMARY net.summary FULL SOURCE VSS VDD
```

“Full” sets the report to go into all subcells, but not add other nets.

**Example 3**

For the sample layout shown in [Figure 4-298](#), if LAYOUT XYZ1 is specified the other optional settings would produce results as shown in [Table 4-48](#).

**Figure 4-298. PEX Report Netsummary Example**



**Table 4-48. Nets Reported for Secondary Keywords**

Secondary Keywords	DETAIL	SUMMARY
<b>FULL</b> (default)	XYZ, A/5-100, A/5-2, A/C/10-100, A/B/10-100, D/6-200, D/6-2, D/C/10-100, D/C/10-100	XYZ, A/5, A/C/10, A/B/10, D/6, D/C/10, D/B/10
<b>LOCAL</b>	XYZ	XYZ
<b>ALL</b>	Full, plus A/C/20-3, A/B/200-4, D/C/20-3, D/B/200-4	Full, plus A/C/20-3, A/B/200-4, D/C/20-3, D/B/200-4

# PEX Report Point2Point

Parasitic extraction

**PEX REPORT POINT2POINT [UNIT LENGTH | DBU] *in\_file* [*out\_file*]  
[LAYOUTNAMES | SOURCENAMES]**

## Summary

Generates a report for point-to-point resistance on specified nets and writes it to a specified output file, or to the screen. A transistor-level extraction is required for generating the report. This statement can be specified multiple times.

## Parameters

- UNIT LENGTH

Optional parameter that sets the units for COORD location parameter in *in\_file* to a user-specified unit. UNIT\_LENGTH uses the value set by the [Unit Length](#) SVRF statement. If Unit Length is not specified, microns is the default.

- DBU

Optional parameter that sets the units for COORD location parameter in *in\_file* to the layout database units.

- *in\_file*

Required name of a file containing entries in the following format for a point-to-point resistance calculation:

```
RESISTANCE net_name location net_name location
```

where:

- RESISTANCE is a required keyword
- *net\_name* is the name of a net
- *location* can take one of the following forms:

PIN <i>device_name</i> <i>pin_name</i>	Name of a device pin
PORT   PROBE <i>p_name</i>	Name of port or probe
COORD <i>x</i> <i>y</i> <i>layer</i>	Coordinates and layer name to be used to find the closest node on the desired net. The coordinates are in database units (DBU) or user specified (UNIT_LENGTH).

The file may contain comments. Comments begin with // and run to the end of the line.

**Note**

PIN, PORT, and PROBE names are from the layout, not from the source. Using source-based names in a PEX Report Point2Point statement is not supported.

---

- *out\_file*

An optional argument naming the output file. The default is standard out. The format for *out\_file* is the same as *in\_file* with the addition of the calculated resistance at the end of the line. The value of the reported resistance is in ohms.

- **LAYOUTNAMES**

An optional keyword that specifies to use the net names from the layout in the report. This is the default behavior.

- **SOURCENAMES**

An optional keyword that specifies to use the net names from the source netlist in the report.

## Description

Generates a report for point-to-point resistance on specified nets and writes it to a specified output file, or to the screen. A transistor-level extraction is required for generating the report.

You can include multiple PEX Report Point2Point statements in the same rule file. The Calibre xRC formatter generates a report for each statement.

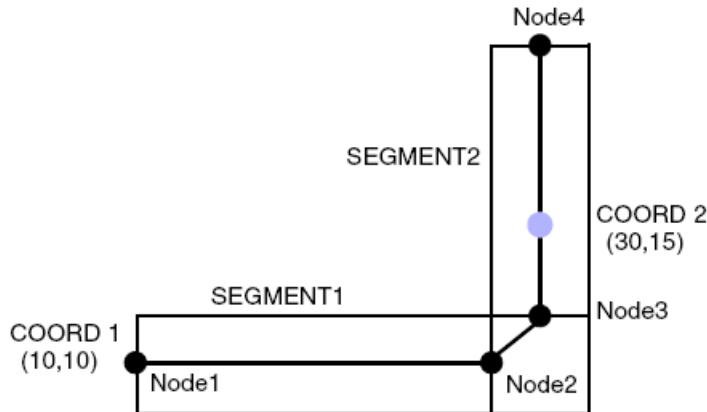
The SOURCENAMES option must be used if any of the specified net names are from the source netlist.

It is possible to mix ports, probes, pins, and coordinates in the same statement. That is, you can specify the starting net and location from a pin and the ending net and location from a coordinate.

When using the COORD option, the terminals of the closest extracted resistor are used. If no extracted resistor exists at one of the coordinates, then the reported value is calculated based on a fraction of an existing parasitic resistor. When this occurs, the Calibre xRC tool issues a warning statement. [Figure 4-299](#) shows an example of this case. A resistance report statement is specified as:

```
RESISTANCE COORD 10 10 M1 COORD 30 15 M1
```

where the coordinates (10, 10) and (30, 15) are in microns.

**Figure 4-299. COORD Usage Example**

COORD 1 is aligned with the start of SEGMENT1, which has an associated parasitic resistance. COORD 2 is not aligned with the start or end of SEGMENT2. PEX Report Point2Point will report the resistance from COORD 1 to COORD 2 by estimating how much of the SEGMENT2 resistance is used. In this case, COORD 2 is at about 1/3 of SEGMENT2. The warning issued by the Calibre xRC tool is as follows:

```
WARNING: For COORD (30, 15) an exact node was not found.  
WARNING: A new node between node 3 and node 4, at 33% from node 3 will be inserted.
```

If the specified locations are not electrically connected, then a large resistance value, 1e38 Ohms, is reported. If the locations do not exist or there are other problems with how the locations are specified, the output from PEX Report Point2Point will contain an error message as follows:

```
RESISTANCE net_name location net_name location: 'ERROR: ...'
```

## Examples

### Example 1

This example shows a point-to-point resistance calculation between the drain pin of a MOSFET to the gate pin of another MOSFET. Since COORD is not specified in the input file, it is not necessary to specify DBU or UNIT\_LENGTH.

```
// SVRF Rule File Statements  
PEX REPORT POINT2POINT myInput.res myOutput.rep  
  
// Point-to-Point Input File  
// <type> <net_name> <location> <net_name> <location>  
RESISTANCE top_only PIN M200 d top_only PIN M300 g
```

where:

- *top\_only* is the net name
- *PIN M200 d* defines the drain pin on MOSFET M200

- o *PIN M300 g* defines the gate pin on MOSFET M300

The output for this statement is:

```
// POINT-TO-POINT REPORT
RESISTANCE top_only PIN M200 d top_only PIN M300 g 9.8822204e+00
```

### Example 2

This example shows a point-to-point resistance calculation between a probe point and a coordinate on the M1 layer using the specified unit length.

```
// SVRF Rule File Statements
UNIT LENGTH u          //microns
PEX REPORT POINT2POINT UNIT_LENGTH myInput.res myOutput.rep

// Point-to-Point Input File
// <type> <net_name> <location> <net_name> <location>
RESISTANCE NET1 PROBE probel NET1 COORD 511 10 M1
```

where:

- o *NET1* is the net name
- o *PROBE* defines a user specified probe point on the *NET1* net, *probel*
- o *COORD* defines a location in the layout on the M1 layer for the *NET1* net

The output for this statement is:

```
// Point-to-Point REPORT
// For the COORD option: 1 database unit = 1000 * 1 user unit
RESISTANCE NET1 PROBE probel NET1 COORD 511 10 M1 1.9117458e-01
```

Since the COORD option is specified, a statement in the output file shows how the unit length is applied.

# PEX Resistance Parameters

Parasitic extraction

## PEX RESISTANCE PARAMETERS *layer* [*layer*]

[TC1 *value*] [TC2 *value*]  
 [MAXLENGTH *length*]  
 [MAXAREA *area*]  
 [BULKRESISTANCE *sheetres*]

### Summary

Specifies layer-specific parameters for resistance calculations.

### Parameters

- ***layer***

A required original layer or a derived polygon layer. The *layer* argument can appear once or twice. When two layers are present, the parameters are applied to vias between the two layers.

- **TC1 *value***

An optional keyword set specifying the first-order temperature coefficient of resistance for the layer. Units are degrees Celsius. The default value is 0.

- **TC2 *value***

An optional keyword set specifying the second-order temperature coefficient of resistance for the layer. Units are degrees Celsius. The default value is 0.

- **MAXLENGTH *length***

An optional keyword set that defines a maximum dimension. The *length* argument is a non-negative real number in units set by [Unit Length](#).

When used with one layer, it is interpreted as the maximum length of the rectangular Manhattan elements on the layer. If this option is not set either with PEX Resistance Parameters, [Resistance Rho](#), or [Resistance Sheet](#), resistors are split every 100 microns. If there is a polygon in the design that is wider than MAXLENGTH, then it is broken into a two dimensional array of parasitic resistors.

When used with two layers, it defines the distance used to control the distribution of reduced vias. The distance between the distributed vias will not exceed this limit.

- **MAXAREA *area***

An optional keyword set specifying an area threshold for large area vias. MAXAREA can only be used with the two-layer format for this statement. Vias with areas greater than this value are modeled as a set of distributed connectors.

- **BULKRESISTANCE *sheetres***

An optional keyword set specifying the sheet resistance of the bulk layer. The *sheetres* parameter is a non-negative, floating-point number specifying the sheet resistance in ohms.

BULKRESISTANCE cannot be used for resistive layers that have specified values for Resistance Sheet, Resistance Rho, or PEX Table NOM\_RSH and PEX Table NOM\_RHO.

### Description

Specifies layer-specific parameters for resistance calculations. The commands that specify maximum element length and temperature coefficients have this order of priority:

1. PEX Table
2. Parasitic Variation
3. PEX Resistance Parameters
4. Resistance Rho
5. Resistance Sheet

PEX Table and Parasitic Variation statements may affect TC1 and TC2. They do not affect MAXLENGTH or MAXAREA.

The temperature coefficients specified by TC1 and TC2 are used with [PEX Thickness EQN](#) to modify extracted resistance. TC1 and TC2 may be overridden based on drawn width using [Parasitic Variation](#) or PEX Table rules.

### MAXLENGTH Keyword

When used with one layer, the MAXLENGTH setting specifies the maximum length of a segment. After fracturing, rectangular conductive shapes aligned along the design's axes on *layer* with an edge longer than *length* are split into smaller rectangles of equal size. These are then represented by a parasitic resistor. Angled and non-rectangular shapes are not split, regardless of size. Generally, 100 microns ensures that interconnect wires are only fractured along their length and not also their width. Setting *length* to a value less than the width of standard interconnect causes excessive fracturing which leads to longer run times.

When MAXLENGTH is used with two layers, it specifies the maximum dimension of the via area in x or y. If either the x or y dimension of the bounding box of a via is greater than MAXLENGTH, the via is cut into segments, each with dimensions less than MAXLENGTH.

For consistent fracturing during extraction, the *area* value of MAXAREA should be set to *length* squared.

### BULKRESISTANCE Keyword

The BULKRESISTANCE setting is used to calculate resistance between the bulk pin of a transistor and the nearest substrate or well connection. The bulk pin of a transistor is normally shorted to a power or ground potential. However, to increase the accuracy of modeling the IR drop to a transistor's bulk pin, specify BULKRESISTANCE to connect the pin to the nearest

substrate or well tap through an extracted resistor. [Table 4-49](#) shows the outcome of extraction when sheet resistance and BULKRESISTANCE are set for bulk or well layers.

**Table 4-49. Specifying Sheet Resistance and BULKRESISTANCE**

Bulk Layer Sheet Resistance	BULKRESISTANCE	Effect
Not specified	Not specified	Bulk pin of MOSFET is connected to an arbitrary point on power or ground nets.
Not specified	Set to 0	Bulk pin of MOSFET is connected to nearest bulk tap.
Not specified	Set to non-zero value	Bulk bin of MOSFET is connect to nearest tap through resistor.
Specified	Not specified	Bulk layer resistance is extracted.
Specified	Set to 0 or greater value	Error Generated.

Transistors with a bulk connection must have a pin named “b” or “bulk.” The bulk pin must be connected to the same net as the nearest bulk tap. For example, if the bulk pin of an NMOS transistor is connected to net VSS:1, then the p-substrate tap must also be connected to VSS:1 to be considered as the nearest connection.

It is possible for bulk or well layers to be serially connected with similar layers through [Connect](#) statements. Multiple layers connected like this can be used to connect a transistor bulk pin to a bulk or well tap. In this case, use BULKRESISTANCE to define all of the layers included in the connection.

Layers that are used in PEX Resistance Parameter BULKRESISTANCE cannot be used in [PEX Ground Layer](#), [PEX Alias](#), [PEX Ignore Capacitance](#), and [PEX Ignore Resistance](#). If these statements use the same layers in the same rule file, Calibre xRC outputs an error message.

For multi-corner extraction, the bulk resistance value does not change. Corners are not applied to the bulk resistance value.

## Examples

### Example 1

These statements show how to use Resistance Sheet to specify resistivity for metal1 layer and PEX Resistance Parameters to add temperature sensitivity.

```
RESISTANCE SHEET metal1 [0.03 80]
PEX RESISTANCE PARAMETERS metal1 TC1 0.003 TC2 0.012
```

**Example 2**

The following lines show an example of extracting resistance for large area vias. The unit length is set to microns.

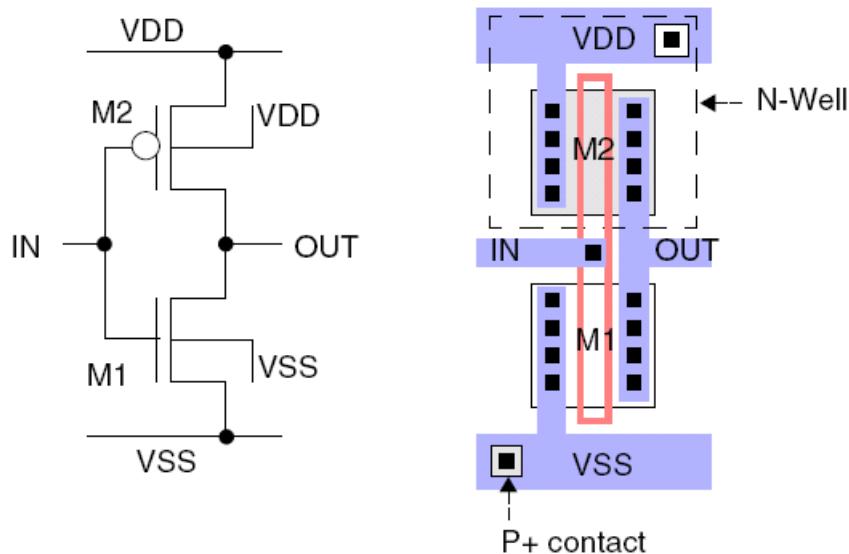
```
UNIT LENGTH U
PEX RESISTANCE PARAMETERS Metal6 Metal7 MAXLENGTH 100 MAXAREA 10000
```

In this example, a via is defined between Metal6 and Metal7. The threshold for x or y dimensions of the via bounding box, as set by MAXLENGTH, is 100 microns. The area threshold for breaking up the via into smaller sections is set by MAXAREA to be 10000 square microns, or MAXLENGTH squared.

**Example 3**

This example shows how to use the BULKRESISTANCE parameter. [Figure 4-300](#) shows an inverter schematic and its layout.

**Figure 4-300. Inverter Cell with Ideal Bulk and N-Well Taps**



The SPICE netlist for this inverter is as follows:

```
* Inverter
* Pin Order: Mxx Drain Gate Source [Bulk] Length Width Model
XM1 VSS IN OUT VSS L=1 W=5 NMOS
XM2 VDD IN OUT VDD L=1 W=5 PMOS
```

The P+ contact on VDD and N+ contact on VSS are typical of a cell-level layout and are considered close power and ground taps to N-Well and p-substrate, respectively. The cross section of the NMOS transistor M1 is shown in [Figure 4-301](#). In this case, it is not necessary to account for a resistance between the substrate tap and bulk pin because the pins are close.

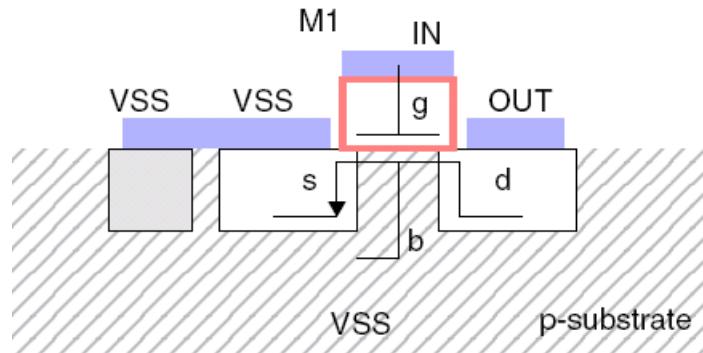
**Figure 4-301. Cross-Section of Ideal PSUB Tap to NMOS Bulk Pin**

Figure 4-302 shows a non-ideal case, where the nearest substrate connection is found in a different cell.

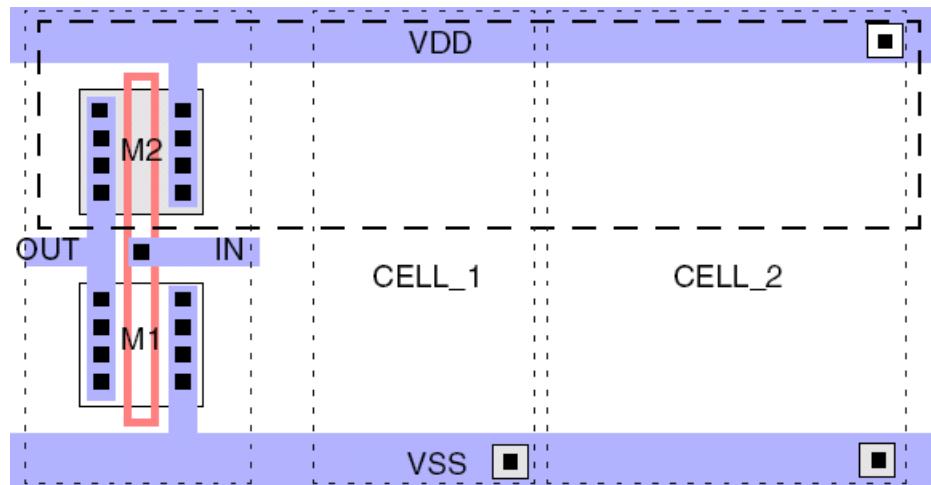
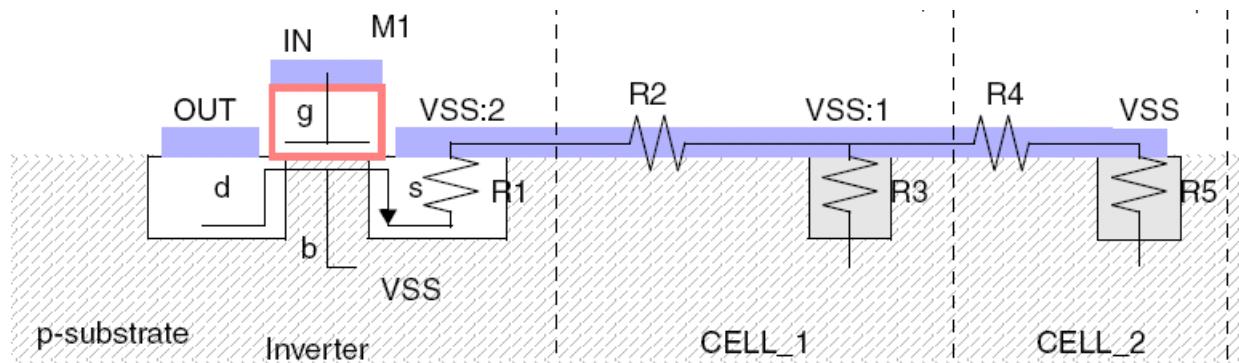
**Figure 4-302. Non-Ideal PSUB Tap Found in CELL\_1**

Figure 4-303 shows the extracted network in close proximity to M1 if the BULKRESISTANCE parameter is not used.

**Figure 4-303. Extracted Network Without Using BULKRESISTANCE**

## PEX Resistance Parameters

The extracted netlist is as follows:

```
* Inverter
XM1 OUT IN VSS:2 VSS L=1 W=5 NMOS
XM2 OUT IN VDD VDD L=1 W=5 PMOS

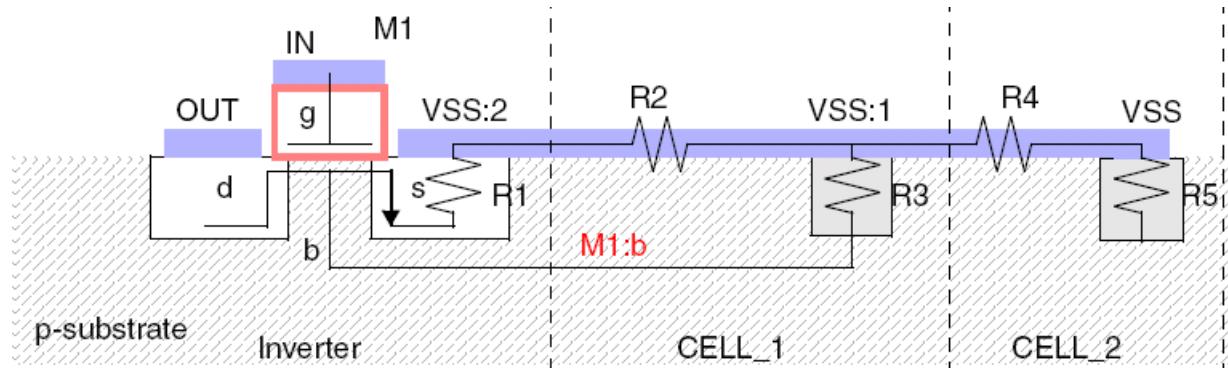
** Parasitic Resistors
** Pin Order: POS NEG Value
R1 VSS:2 M1:s 1
R2 VSS:2 VSS:1 2
* R3 is considered a dangling resistor, will not appear in netlist
R4 VSS:1 VSS 2
* R5 is considered a dangling resistor, will not appear in netlist
```

To connect the M1 bulk pin and the nplus substrate contact in CELL\_1, include the following SVRF statement in the rule file:

```
PEX RESISTANCE PARAMETERS psub BULKRESISTANCE 0
```

Figure 4-304 shows that the bulk pin of M1 is shorted to R3 (nearest substrate tap) through the net M1:b.

**Figure 4-304. Extracted Network with BULKRESISTANCE set to 0**



The extracted netlist is as follows:

```
* Inverter
XM1 OUT IN R1:NEG XM1:b L=1 W=5 NMOS
XM2 OUT IN VDD XM2:b L=1 W=5 PMOS

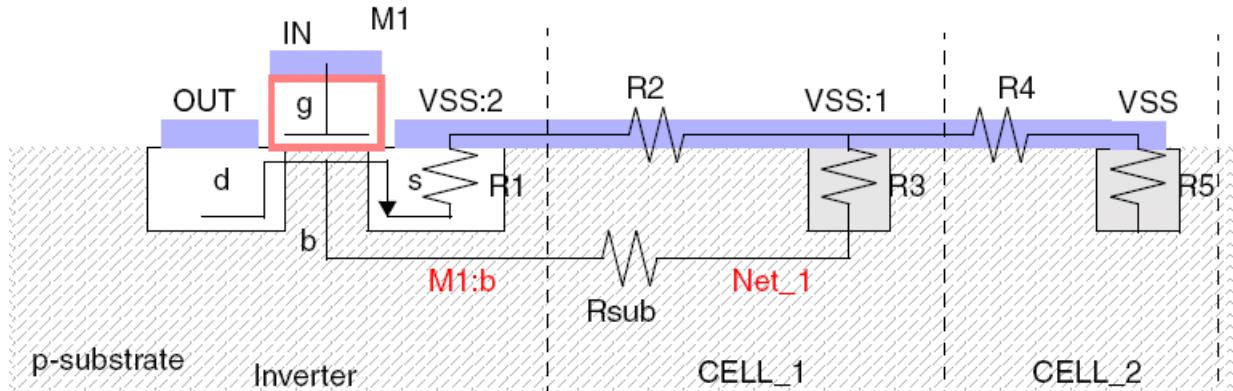
* Parasitic Resistors
R1 VSS:2 XM1:s 1
R2 VSS:2 VSS:1 2
R3 VSS:1 XM1:b 1
R4 VSS:1 VSS 2
* R5 is considered a dangling resistor, will not appear in netlist
```

To include substrate resistance between the bulk pin and substrate contact, include the following SVRF statement in the rule file:

```
PEX RESISTANCE PARAMETERS psub BULKRESISTANCE 109.1
```

Figure 4-305 shows how the extracted resistor is connected between the M1 bulk pin and the nearest nplus substrate tap in CELL\_1.

**Figure 4-305. Extracted Network with Non-Zero BULKRESISTANCE**



The extracted netlist is as follows:

```
* Inverter
XM1 OUT IN R1:NEG XM1:b L=1 W=5 NMOS
XM2 OUT IN VDD XM2:b L=1 W=5 PMOS

* Parasitic Resistors
R1 VSS:2 XM1:s 1
R2 VSS:2 VSS:1 2
R3 VSS:1 Net_1 1
R4 VSS:1 VSS 2
Rsub XM1:b Net_1 36
* R5 is considered a dangling resistor, will not appear in netlist
```

## PEX Sensitivity

Parasitic extraction

### PEX SENSITIVITY {ON | OFF}

[REFERENCE *center*]

[VARIATION *layer* [VARIATION *layer* ...]]

[TEMPERATURE]

### Parameters

- **ON**

Required keyword that switches on extraction with sensitivity. ON is the default behavior when the statement is not included.

- **OFF**

Required keyword that switches off extraction with sensitivity.

- **REFERENCE *center***

An optional keyword set that specifies the foundry-supplied process corner to use as the center for the sensitivity calculations. The default value for *center* is the typical process corner.

- **VARIATION *layer***

Optional keyword set that specifies layers used for sensitivity variations. More than one layer maybe specified.

- **TEMPERATURE**

Optional keyword that specifies sensitivity extraction using temperature effects.

### Description

Specifies settings for generating a netlist with process variation sensitivity information using one of the process corners as reference. Process variations include metal thickness, line width, and inter-layer dielectric (ILD) thickness.

PEX Sensitivity supports generating sensitivity SPEF (S-SPEF) and ELDO netlists with sensitivity coefficients for process variation parameters. The value of a parasitic device corresponds to a set of process variation parameter values.

When the TEMPERATURE keyword is used, temperature sensitivity coefficients are calculated during the formatting step (-xrc -fmt) from TC1 and TC2 which are specified in the [PEX Extract Temperature](#) statement.

### Examples

This example uses the typical corner as reference and defines variations in metal 1 to metal 3:

```
PEX SENSITIVITY ON
VARIATION metal1
VARIATION metal2
VARIATION metal3
```

## PEX Skin Include

Parasitic extraction

**PEX SKIN INCLUDE** [LAYOUT | SOURCE] *net\_name* [net\_name ...]

---

**Note**

 PEX Skin Include has been renamed to [PEX Inductance Skin Include](#). There are no changes to parameter definitions and functionality. PEX Skin Include has been deprecated as of the 2009.1 release.

---

# PEX Slots Handling

Parasitic extraction

**PEX SLOTS HANDLING [layer]**{**OFF** | **COUNT\_THRESHOLD** *count* [**AREA\_RATIO** *area*]  
[**SHAPE\_FACTOR** *factor*]}**Parameters**

- ***layer***

An optional keyword that specifies a layer name. If *layer* is not specified, the statement will be applied to all layers.

- **OFF**

A required keyword that switches off the statement. This is the default behavior.

- **COUNT\_THRESHOLD *count***

A required keyword set that specifies identifying slotted polygons when the number of holes in a polygon exceeds *count*. The *count* parameter is a non-negative integer value. A *count* of zero is equivalent to off.

- **AREA\_RATIO *area***

An optional keyword set that specifies the maximum percentage of slots that may occupy a polygon. If the slots account for more than the *area* percentage of the total area, the polygon is not considered slotted. The default value is 0.3.

- **SHAPE\_FACTOR *factor***

An optional keyword set that specifies the correction factor for slotted polygon resistivity. This adjustment factor is used to account for non-square slots in a polygon and adjusts the sheet resistance value used for calculating the parasitic resistance. The adjustment is calculated as:

$$\frac{1}{1 - \text{factor} \times \sqrt{\frac{\text{SlotsArea}}{\text{TotalArea}}}}$$

The default value for *factor* is 1.0.

**Description**

Specifies how to control extracting parasitics from slotted layers. Calibre xRC does not differentiate slotted polygons from non-solid polygons. Use PEX Slots Handling in a rule file to control how slotted polygons are modeled for extraction.

**Note**

For more information on how to handle layers with slotted polygons, see “[Modeling Slotted Metal](#)” in the *Calibre xRC User’s Manual*.

## Examples

The following example sets up slot handling for the metal 1 layer:

```
PEX SLOTS HANDLING METAL1 COUNT_THRESHOLD 15 AREA_RATIO 0.5
```

In this example, metal1 polygons must have a minimum of 15 slots to qualify as slotted metal and the slots must occupy less than 50% of the polygon area. The SHAPE\_FACTOR default of 1.0 is used.

## PEX Temperature

Parasitic extraction

**PEX TEMPERATURE  $T$  [NOMINAL  $T_n$ ] [RANGE  $T1 T2 T3$ ]**

---

**Note**

 PEX Temperature has been renamed to [PEX Extract Temperature](#). There are no changes to parameter definitions and functionality. PEX Temperature has been deprecated as of the 2008.3 release.

---

# PEX Thickness EQN

Parasitic extraction

## PEX THICKNESS *layer* EQN [RONLY]

### Parameters

- *layer*  
A required original layer.
- RONLY  
An optional keyword indicating that the statement applies to thickness calculations for resistance only.

### Description

#### Note



These values are set based on foundry data. Changing the value will affect accuracy. Do not modify this statement in your rule file.

This statement is output by the xCalibrate rule file generator, and specifies the thickness equation as a function of drawn width and local density. The commands that specify thickness calculation have this order of priority:

1. PEX Thickness EQN
2. Parasitic Variation
3. PEX Contact Capacitance when PEX Thickness EQN is present in the rule deck.
4. PEX Density Window when PEX Thickness EQN is present in the rule deck.
5. PEX Thickness Nominal

If the RONLY form is present in the rule file, there must also be a form of the PEX Thickness EQN statement for the layer without RONLY to be used for capacitance calculations.

PEX Thickness EQN must be followed by a pair of square brackets ( [ ] ) enclosing the thickness calculation equation. The brackets must include all statements, equations, and functions related to calculating thickness. The equation must conform to the following rules:

- The density variable is *density()*.
- The drawn width variable is *width()*.
- The equation can include:
  - Intermediate variables (for example; K1, K2, D1).
  - Intermediate equations.
  - IF - ELSE statements.

- All C++ math.h functions.
- The equation must *not* include:
  - Looping statements, such as WHILE, DO, FOR.
  - Conditional statements, such as SWITCH.

For a description of the Calibre xRC in-die variation process, including information on when and when not to use it, refer to the [\*Calibre xRC User's Manual\*](#).

## Examples

```
PEX THICKNESS metall EQN
[
    PROPERTY T
    if (width() <= 0.1) {
        a=2.0*width()
    } else {
        a=pow(width())
    }
    T=a * pow(density(),2.0)
]
```

# PEX Thickness Nominal

Parasitic extraction

**PEX THICKNESS *layer* NOMINAL *value***

## Parameters

- ***layer***  
A required original layer.
- ***value***  
A required keyword that specifies the nominal thickness value in user units.

## Description

### Note



These values are set based on foundry data. Changing the value will affect accuracy. Do not modify this statement in your rule file.

This statement is output by the xCalibrate rule file generator, and specifies the nominal thickness for a layer. Used with in-die variation calculations.

### Caution



1. [PEX Thickness EQN](#) overrides calculation of thickness using PEX Thickness Nominal.
2. [PEX Density Window](#) overrides PEX Thickness Nominal, if [PEX Thickness EQN](#) is present in the rule deck.
3. [PEX Contact Capacitance](#) overrides PEX Thickness Nominal, if [PEX Thickness EQN](#) is present in the rule deck.

For a description of the Calibre xRC in-die variation process, including information on when and when not to use it, refer to the [Calibre xRC User's Manual](#).

## Examples

```
PEX THICKNESS poly NOMINAL 0.173000
```

## PEX Threshold

Parasitic extraction

**PEX THRESHOLD**  $\{\{R \mid C \mid D\} constraint\}$ 

Used in Calibre xRC.

**Note**As of version 2007.3, this statement is deprecated. Use [PEX Reduce Digital](#) instead.

### Parameters

- **R constraint**

Specifies the threshold is based on the sum of Rs in the net model. The **constraint** is scaled by the [Unit Resistance](#) statement's value. The default value is 1.0 ohm.

- **C constraint**

Specifies the threshold is based on the sum of Cs in the net model. The **constraint** is scaled by the [Unit Capacitance](#) statement's value. The default is 1e-12 farads.

- **D constraint**

Specifies the threshold is based on the product (Total R) \* (Total C). The **constraint** is scaled by the [Unit Time](#) statement's value. The default is 1e-9 seconds.

See the “Constraint Notation” column of [Table 2-2](#) on page 45.

### Description

Specifies thresholds for distributed RC parasitic extraction for nets such that if the specified threshold constraint is met, then the Calibre xRC tool will output distributed RC and/or lumped C parasitic results for these nets to the netlist.

Thresholds decrease the size of netlists and databases, resulting in faster simulations. This is because lumped results in general are smaller and process faster. For example, you could specify a resistance threshold of 10 if you know that the accuracy of the resistance value for nets smaller than 10 resistance units is not significant.

For Calibre xRC, the format stage converts distributed RC models that do not meet the threshold value into lumped C models by discarding the resistors. When you perform lumped C extraction, then the tool lumps the distributed RC models by definition, so thresholding does not apply.

You generally use the greater than (>) constraint in this statement. Multiple thresholds can be specified using a single statement, in which case, the constraint is met if and only if all constraints are TRUE.

## Examples

Suppose you specify the following statement in the rule file:

```
PEX THRESHOLD R > 10
```

The tool outputs all the distributed RC parasitic models that have total resistance of 10 and above. Whenever it finds a parasitic model with total R less than 10, it discards the resistors to convert them into lumped C models. That is, if the constraint is not met for a given parasitic model, the PDB stage outputs lumped C results.

# PEX Tolerance Distributed

Parasitic extraction

## PEX TOLERANCE DISTRIBUTED {R *value* | RC *value*} [CONNECTION]

Used in Calibre xRC.

---

**Note**

As of version 2007.3, this statement is deprecated. Use [PEX Reduce Digital](#) instead.

---

## Parameters

- **R *value***

A required keyword set that specifies to base reduction on resistor values below the specified tolerance ***value***.

- **RC *value***

A required keyword set that specifies to base reduction on (total capacitor \* resistor) values below the specified tolerance ***value***.

- **CONNECTION**

An optional keyword that specifies to preserve all layers, even if the resistance of a layer is below the specified tolerance. This statement is generally used to prevent vias and contacts from being reduced into other layers.

## Description

Specifies to run bulk reduction during distributed RC or R-coupled C parasitic extraction such that the resistance and capacitance along the nets are preserved.

This specification statement performs geometry-based reduction. It reduces nets so that they are electrically and topologically equivalent, which may be important for some analysis tools. This reduction method differs from that performed with the [PEX Reduce Digital](#) statement. It reduces the network without complex analysis, thus generating more predictable, easily mappable netlists.

Calibre xRC does not reduce protected nodes, as follows:

- Nodes that connect more than two resistors, which preserves branch points in the circuit.
- Nodes that contain an hprobe, device pins, or port, which preserves I/Os.
- Optionally, via and contact nodes, which preserves the layer structure.

For R-based reduction, Calibre xRC lumps all unprotected resistors of resistance less than the specified ***value*** into a neighboring larger resistor. Associated capacitors are split equally between two neighboring resistors.

For RC-based reduction, Calibre xRC first calculates the total capacitance of an unprotected node, which is the sum of the intrinsic and coupled capacitances. It then multiplies it by the

resistance value. If the result is less than the specified *value*, Calibre xRC lumps the resistance into a neighboring larger resistor. Intrinsic capacitors are split equally between two neighboring resistors, however coupled capacitors are moved in the direction of the larger resistor.

## Examples

PEX TOLERANCE DISTRIBUTED R 10 CONNECTION

## PEX Via Capacitance

Parasitic extraction

**PEX VIA CAPACITANCE {YES | NO} [DIAGONAL]**

### Parameters

- **YES**  
A keyword that includes via effects in capacitance calculations. This is the default.
- **NO**  
Disables via capacitance calculations. Must be specified to disable this feature.
- **DIAGONAL**  
An optional keyword that adds via capacitance calculations for vias diagonally located from each other.

### Description

This optional statement can be used to turn off the calculations for all via capacitance. When used with [PEX Contact Capacitance](#), these statements control contact and via extraction. If neither statement appears in a rule file, then via and contact capacitances are extracted.

### Examples

To make the default behavior explicit in your rule file, add the following line:

```
PEX VIA CAPACITANCE YES
```

To exclude vias and contacts from extraction, use the following statement:

```
PEX VIA CAPACITANCE NO
```

If both of the following statements are present in the rule file, then only vias will be extracted:

```
PEX VIA CAPACITANCE YES  
PEX CONTACT CAPACITANCE NO
```

To calculate capacitance for contacts only, include the following statements in the rule file:

```
PEX VIA CAPACITANCE NO  
PEX CONTACT CAPACITANCE YES
```

To include capacitance for vias located horizontally and diagonally from each other, include the following statement in the rule file:

```
PEX VIA CAPACITANCE YES DIAGONAL
```

## PEX Via Reduction Resistance

Parasitic extraction

**PEX VIA REDUCTION RESISTANCE** [*layer1 layer2*]

{**STANDARD** [*COUNT max*] [*DISTANCE value*] |

**FLEXIBLE** [*DISTANCE value*] |

**OFF**}

---

**Note**

 PEX Via Reduction Resistance has been deprecated as of the 2008.3 release. This statement has been renamed to [PEX Reduce Via Resistance](#). There are no changes to parameter definitions and functionality.

---

## PEX Xcell

Parasitic extraction

**PEX XCELL** *layout\_cell\_name* [CONTEXT *layout\_path* | PCDEF [INTRINSIC] | PRIMITIVE [GDS *file.gds*] | IDEAL]

### Parameters

- ***layout\_cell\_name***

A required parameter that specifies the layout name of the xcell. Only one *layout\_cell\_name* can be specified per statement. It may contain one or more asterisk (\*) characters; the \* character is a wildcard that matches zero or more characters. A string that contains a \* character must be enclosed in quotation marks because the \* is a reserved symbol.

- CONTEXT *layout\_path*

An optional keyword and parameter set used for in-context extraction. For more information on in-context extraction, see “[In-Context Extraction](#)” in the *Calibre xRC User’s Manual*.

The *layout\_path* is any valid layout path or “parent\_id/cell\_id”. See “[Discovering Layout Paths for In-Context Cells](#)” in the *Calibre xRC User’s Manual* to determine a valid *layout\_path*.

If doing gate-level extraction the cell is treated as a primitive and the context ignored.

- PCDEF [INTRINSIC]

An optional keyword that indicates the cell is a pcell. The pcell contents are not extracted. The intentional device represented by the pcell will be netlisted. The parasitics outside the pcell boundary will be extracted. The coupling capacitances between nets outside the pcell and the pcell pins will be extracted.

INSTRINSIC — An optional keyword specified with PCDEF that causes the pin intrinsic capacitance to be ignored for the device layers within the cell.

- PRIMITIVE [GDS *file.gds*]

An optional keyword that indicates the cell is a primitive. The contents are not extracted. In gate-level extraction, all xcells are treated as primitives.

GDS *file.gds* — An optional keyword and parameter set specified with PRIMITIVE that indicates where to find the GDS information. May only be used with LEF/DEF designs.

- IDEAL

An optional keyword that indicates the cell is an ideal xcell. The contents of the cell are not extracted, but are written to the netlist.

### Description

This statement specifies a hierarchical extraction cell also referred to as an xcell. This statement allows xcells to be specified in the SVRF file and may be used in combination with an xcell list as long as the specified cell does not conflict with an xcell already defined in the xcell list. For

example, if an xcell is defined for in-context extraction with two different layout paths, this causes a conflict. If a conflict is detected, then extraction stops with an error message. If an xcell is specified using PEX XCELL and also appears in the xcell file, the resulting conflict may be resolved by using the [PEX Xcell Precedence](#) to indicate which definition should take precedence. When performing LEF/DEF extraction, the default xcell list file generated by Calibre will have the lowest precedence and will not trigger warnings or errors.

## Examples

The following is an example of how to specify xcells in your SVRF rule file:

```
PEX XCELL NOR IDEAL
PEX XCELL NAND PRIMITIVE
PEX XCELL NMOS PCDEF
PEX XCELL INV PCDEF INTRINSIC
```

## PEX Xcell Extract Mode

Parasitic extraction

### PEX XCELL EXTRACT MODE {GRAY | BLACK}

#### Parameters

- **GRAY**

Required keyword that specifies using the geometries inside cells in the xcell file during extraction. This is the default behavior.

- **BLACK**

Required keyword that specifies ignoring geometries inside cells that are listed in the xcell file.

#### Description

Used in Calibre xRC for gate-level or LEF/DEF extraction. Specifies how to account for the geometries inside xcells when calculating parasitics.

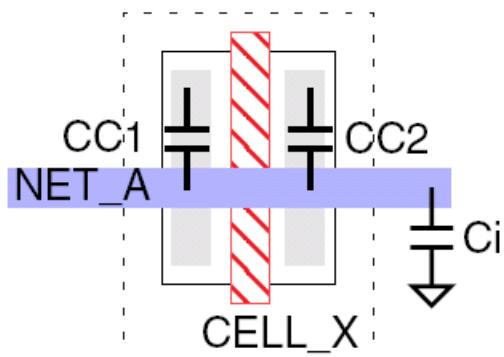
GRAY is the default setting. It indicates that cells contain complete layout information. Use the GRAY setting if you want to include obstruction data for parasitic capacitance calculations in LEF/DEF. For more information on how to handle obstructions, see [PEX DEF Extract Cell Obstructions](#).

Use BLACK when using LEF/DEF layout data as input; that is, when using the abstracted version of standard cells. Standard cell abstracts include pin locations and may also include routing obstructions. When PEX Xcell Extract Mode BLACK is used, only the top cell routing is used for parasitic extraction and internal contents of the cells in the xcell list are ignored.

When using GDS and Milkyway layout data in ASIC mode or using flat extraction, the resulting parasitics include the effects of the standard cell contents.

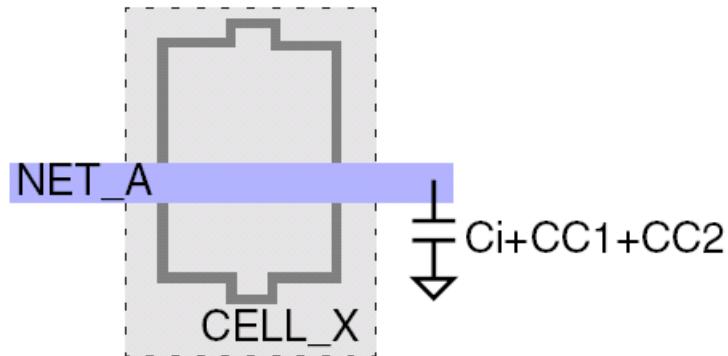
[Figure 4-306](#) shows how the contents of a standard cell are seen during flat transistor-level extraction. CC1 and CC2 are parasitic coupling capacitors between metal in the CELL\_X and top-level interconnect NET\_A. CC1 and CC2 are extracted and netlisted.

**Figure 4-306. Parasitic Extraction with Transistor-Level View of Standard Cell**



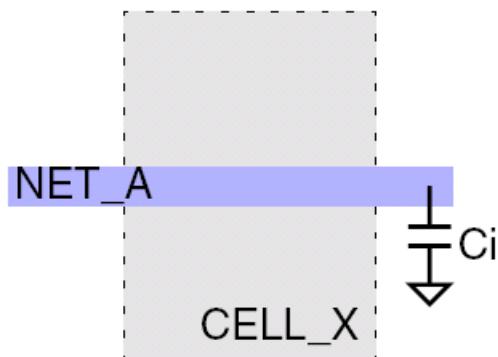
[Figure 4-307](#) shows how the extraction for the top-level routing is modified when PEX XCell Extract Mode GRAY is used. The coupling capacitances from CELL\_X are lumped with the intrinsic parasitic capacitor,  $C_i$ , on NET\_A.

**Figure 4-307. Parasitic Extraction with Gray Box Mode**



[Figure 4-308](#) show how the extraction for the top-level routing is modified when PEX Xcell Extract Mode BLACK is used. No geometries from CELL\_X are included, so no parasitics capacitance from CELL\_X is added to the final parasitic capacitance on NET\_A.

**Figure 4-308. Parasitic Extraction with Black Box Mode**



## Examples

To use the BLACK setting with ASIC extraction include the following line in the rule file:

```
PEX XCELL EXTRACT MODE BLACK
```

## PEX Xcell Precedence

Parasitic extraction

### PEX XCELL PRECEDENCE {STATEMENT | FILE} [BEST]

#### Parameters

- **STATEMENT**

A required keyword that gives higher precedence to the xcell definitions specified in the SVRF rule file with [PEX Xcell](#).

- **FILE**

A required keyword that gives higher precedence to the xcell definitions specified in the xcell list file.

- **BEST**

An optional keyword that allows only the xcell specifications with the most explicit name match to be applied to the corresponding cell.

#### Description

This statement specifies whether the SVRF rule file or xcell list file takes precedence if conflicting xcell definitions are encountered. You can specify this statement once in your rule file. For a detected conflict, a warning is issued indicating the name of the cell and the xcell flags applied. When this statement is not specified, conflicting xcell definitions will result in an error and extraction will stop.

Use the optional keyword BEST for resolving conflicts in cases where multiple wildcard xcell specifications may exist in the file. When a cell name matches multiple xcell definitions with wildcards, this keyword causes the best matching xcell specifications to be applied to the cell.

The following scenarios describe how the BEST keyword determines the best match:

- **Longest name match** — The cell named pmos\_nand2 matches the following xcell names with wildcards:

```
pmos* -P -I  
pmos_nand* -P
```

Applying the BEST option selects the specification for the name pmos\_nand\*. The tool designates pmos\_nand2 as a primitive, but not ideal. The specification for pmos\* is ignored.

- **Least number of wildcards** — The cell named pmos\_nor2 matches the following xcell names with wildcards:

```
pmos*nor* -P  
pmos_nor* -PCDEF
```

Applying the BEST option selects the specification for pmos\_nor\*. The tool designates pmos\_nor2 as PCDEF, but not primitive. The specification for pmos\*nor\* is ignored.

- Same length and same number of wildcards** — The cell named pmos\_xor4 matches the following xcell names with wildcards:

```
pmos_*or4 -P
pmos_xor* -I
```

Applying the BEST option selects the specification for pmos\_\*or4 and pmos\_xor\*. The tool designates pmos\_xor4 as ideal and primitive. If the two specifications conflict, an error is issued and extraction will stop.

The tool keeps the best match local to the file containing the wildcard name. This means the best matched xcell specification is used within the rule file, and the best matched xcell specification is used within the xcell list file. The single best matched xcell name is not applied across both the rule file and xcell list file.

## Examples

### Example 1

To ensure the xcells specified in your SVRF rule have higher precedence, include the following line in the rule file:

```
PEX XCELL PRECEDENCE STATEMENT
```

Any conflicts encountered will issue a warning indicating the name of the cell and the xcell flags applied. For example, if you have the following statement in your rule file:

```
PEX XCELL NAND IDEAL
```

and the following xcell definition in your xcell list file, xcell\_list:

```
NAND NAND -PCDEF
```

the following warning will be issued and NAND will be treated as an ideal xcell:

```
WARNING: Ignoring conflicting xcell specification for cell "NAND"
identified as "PCDEF [previously Ideal]" in file: xcell_list.
```

### Example 2

To give higher precedence to the xcell list file and use the longest matched wildcard names, include the following statement in your extraction rule file.

```
PEX XCELL PRECEDENCE FILE BEST
```

Assume your xcell list file contains the following:

```
pmos_rf* -I -P          // Treat as ideal xcell and primitive
pmos* -P                 // Treat as primitive only
nmos_rf* -P              // Treat as primitive
nmos* -I -P              // Treat as ideal xcell and primitive
```

and your rule file contains the following:

```
PEX XCELL pmos* IDEAL
PEX XCELL pmos_* PRIMITIVE IDEAL
PEX XCELL nmos* PCDEF
```

```
PEX XCELL nmos_* PCDEF INTRINSIC
```

and you have cells pmos\_rf1 and nmos\_rf1. By specifying the options FILE and BEST, the following occurs:

- The FILE option means the xcell list file is processed first.
- The following specifications are applied from the xcell list file:
  - pmos\_rf\* -I -P
  - nmos\_rf\* -Pbecause the names pmos\_rf\* and nmos\_rf\* have the longest xcell name with a wildcard. The cell pmos\_rf1 is treated as primitive and ideal. The cell nmos\_rf1 is only treated as primitive.
- The specifications in the rule are applied after those in the xcell list file. In this case, the best matches are:
  - PEX XCELL pmos\_\* PRIMITIVE IDEAL
  - PEX XCELL nmos\_\* PCDEF INTRINSICwhich does not result in any conflict for pmos\_rf1.
- The specifications given for nmos\_\* and nmos\_rf\* conflict and a warning is generated. Since the FILE option is specified, the xcell definitions in the xcell list file take precedence over the specifications found in the rule file. This means the nmos\_rf1 cell is treated as primitive and not PCDEF intrinsic.

# Pins

Layer operation

## PINS *layer*

Used only in Pyxis Layout.

### Parameters

- *layer*

A required original layer or layer set.

### Description

Generates a derived polygon layer consisting of all the pin shapes on *layer*. Selects the pin shapes from all placements that present only their external aspect to the \$check\_drc() function.

In practice, this means the hierarchy control buttons of the \$check\_drc dialog box in Pyxis Layout are important for determining the output of the Pins operation.

- If the **flat** button is selected, you get no output.
- If the **top** button is selected, you get pin shapes from instances on the top level of the hierarchy.
- If the **peeked** button is selected, you get pin shapes from (the ports of) instances of child cells of the peeked cell.

### Examples

```
rule { PINS poly }
```

# Polygon

Specification statement

**POLYGON** *x1 y1 x2 y2 [xN yN ...] layer*

## Parameters

- *x1 y1 x2 y2*

A set of required floating-point numbers that specifies the coordinates of vertices of a polygon in user units. Exactly two pair of coordinates are interpreted as the lower-left and upper-right corners (they must be specified in that order) of a rectangle having sides parallel, respectively, to the coordinate axes. You can specify *x* *y* parameters any number of times in one statement. You can specify *x* and *y* values in a [Variable](#) statement.

- *layer*

A required original layer or layer set name. If it is a simple layer name, this is equivalent to the use of its layer number in the statement. If *layer* is a layer set name, this is equivalent to the statement being repeated for each element of the layer set. The layer does not have to be in the input layout database.

## Description

Defines a polygon having the given (*x*, *y*) coordinates on the given layer. A polygon defined by this statement behaves exactly as if it were present in the database on the given layer.

Two *x* *y* coordinate pairs are the minimum number of pairs that generate a polygon. If more than two *x* *y* coordinate pairs are specified, then the first and last pairs are assumed consecutive points of the polygon and do not need to coincide. The polygon can be oriented either clockwise or counter-clockwise. The specified polygon must be simple; that is, it cannot self-intersect.

Multiple Polygon specification statements are allowed.

Polygons from this statement are generated at the specified coordinates in user units regardless of changes to the rule file [Precision](#). Polygons generated by this statement are affected by [Layout Magnify](#) with an explicit *value*; the **AUTO** keyword has no effect, however.

This statement is not used by Direct mode ICVerify applications.

See also [Layout Polygon](#).

## Examples

This example shows how to use Polygon to create an exclusion area layer.

```
LAYER CNT 7      // All contacts
LAYER M1 8 36   // All metal1
LAYER M2 14 37 // All metal2
LAYER VA 13     // All vias
...
// Geometry inside the rectangle [-8568,-8924.4] to [9548.2,9728]
// is not to be checked.
LAYER CHECK_MASK 100
POLYGON -8568 -8924.4 9548.2 9728 CHECK_MASK
```

```
...
CONTACT = CNT NOT CHECK_MASK // Contacts to check
MET1 = M1 NOT CHECK_MASK      // Metall1 to check
...
```

## Port Depth

Specification statement

### **PORT DEPTH {PRIMARY | ALL | *number*}**

Used only in Calibre nmLVS/nmLVS-H and Calibre xRC.

#### Parameters

- **PRIMARY**

Keyword that instructs the tool to use free-standing port objects from only the top-level cell. This is the default behavior if you do not include this statement in the rule file.

- **ALL**

Keyword that instructs the tool to use free-standing port objects from throughout the hierarchy.

- ***number***

Non-negative integer, variable, or numeric expression, which instructs the tool to use free-standing port objects from *number* levels below the top-level cell. Specifying zero is equivalent to PRIMARY.

#### Description

Specifies hierarchical depth for reading port objects from the geometric layout database for use in the top-level cell. This statement supports connectivity extraction only; it may be specified once in your rule file. It operates on port objects specified in the [Port Layer Polygon](#) and [Port Layer Text](#) statements. It does not influence text objects used by the [With Text](#) operation.

Port objects that come from lower levels of the hierarchy are transformed to the top-level coordinate space and are replicated according to the hierarchical structure of the design. Such port objects then behave as if they originated at the top level; this is true in flat as well as hierarchical applications.

In flat applications, only those database port objects that are selected by this statement are used. In hierarchical applications, port objects from all levels of the design hierarchy are used locally in the cells where they appear, regardless of the Port Depth specification statement. Port objects selected by this statement serve as top-level ports in addition to any local role that they may perform.

This statement controls port objects read from geometric databases and those entered with [Layout Text](#) or [Layout Text File](#) specification statements (which behave just as if they were database text). Reading of text ports does not depend on [Text Layer](#) or [Text Depth](#) statements, and reading of polygon ports does not depend on whether the layer is referenced by other operations.

## Examples

### Example 1

Basic syntax.

```
PART DEPTH 0
PART LAYER TEXT port_txt
ATTACH port_txt metal7

CONNECT metal7 metal6 BY via6
...
```

### Example 2

Using variables.

```
VARIABLE level ENVIRONMENT
// level is declared as a numeric variable in the environment

PART DEPTH level
PART LAYER TEXT port_txt
ATTACH port_txt metal7
```

## Port Layer Polygon

Specification statement

**PORT LAYER POLYGON** *layer* [*layer* ...]

Used only in Calibre nmLVS/nmLVS-H.

### Parameters

- *layer*

An original layer or layer set. You can specify *layer* any number of times in one statement.

### Description

Causes shapes on the specified layer(s) to be treated as geometric ports. This statement supports layout ports for geometric input databases only. The port layer is the shape's layer and the port location is the center of the shape's extent if the center falls inside the polygon's area or on its boundary; otherwise, an arbitrary vertex of the polygon is chosen. The port polygon itself is unnamed.

---

#### Note



1. Shapes defined in the rule file (using [Polygon](#) or [Layout Polygon](#) specification statements) do not become ports.
  2. Reading of polygon ports does not depend on whether *layer* is or is not referenced by other operations. However, polygon ports must be attached to polygons of another layer that can actually form ports by using either the explicit ([Attach](#) statement), implicit ([Connect](#) statement), or free ([Label Order](#) statement) method of attachment.
  3. Unless [Layout Merge On Input](#) YES is specified, port polygons use unmerged data with centers computed after path expansion.
  4. Port polygons are flagged for non-orientable and non-simple objects, but do not participate in acute, skew, or offgrid flagging (unless the specified layer is referenced by other operations that cause such flagging). See [Flag Acute](#), [Flag Nonsimple Path](#), [Flag Nonsimple Polygon](#), [Flag Offgrid](#), and [Flag Skew](#).
- 

Reading of port objects into the top-level cell is controlled by the [Port Depth](#) specification statement. Port objects from the specified depth are transformed to top-level space and replicated according to the hierarchical structure of the design. Port objects from lower levels of the hierarchy behave as if they were at the top level. This is true in both flat and hierarchical applications.

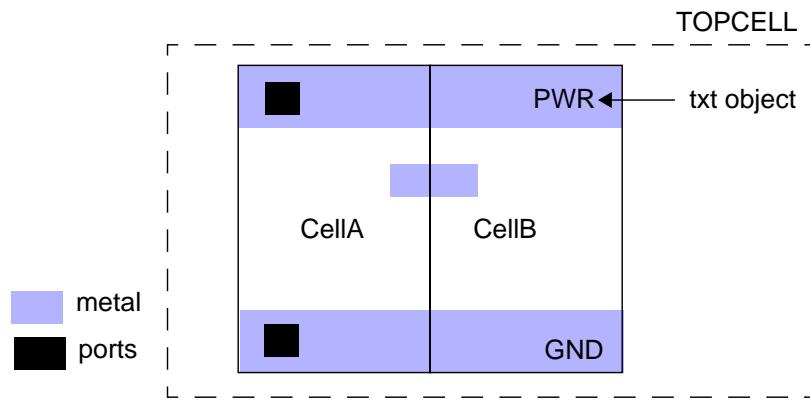
In hierarchical applications, port polygons in the top-level cell are used by the hierarchical SPICE netlister (calibre -spice) to name top-level subcircuit pins. Port names are the same as the respective net names to which the port polygons are attached. If the nets do not have user-given names, then net numbers are used. Port polygons at lower levels of the hierarchy are not used to specify cell pin names. They can, however, be used by PEX applications.

Port polygons at any level of the hierarchy can overlap shapes in lower levels of the hierarchy and can be attached to those lower-level shapes. The hierarchical connectivity extractor forms

any pins in the lower-level cells that are necessary to connect to port objects higher up. This is similar to how layout text is handled hierarchically.

[Figure 4-309](#) shows a layout example.

**Figure 4-309. Port Layer Polygon**



Given the layout objects above, suppose you have the following in the rule file:

```
LAYER metal 10
LAYER txt 40
LAYER ports 50

TEXT LAYER txt           // assign net names from this layer
ATTACH txt metal

PORT LAYER POLYGON ports // designate nets as ports using this layer
ATTACH ports metal
```

You would see GND and PWR ports at the top level as follows:

```
.SUBCKT TOPCELL GND PWR $$ net names appear as port names due to
$$ PORT LAYER POLYGON objects
** N=4 EP=1 IP=8 FDC=6
X0 3 GND PWR CellA $T=0 0 0 0 $X=-4250 $Y=0
X1 3 GND PWR CellB $T=29000 0 0 0 $X=24500 $Y=0
```

For additional information, see “Netlisting of Ports and Pins” in the [Calibre Verification User’s Manual](#).

See also [Port Layer Text](#).

### Examples

In this example, top-level metal is texted. To have top-level ports recognized, port polygons from layer 50 are placed on top-level polygons from the top-level metal layer.

```
...
LAYER metal7 27
LAYER txt 30
LAYER ports 50

...
// in nmLVS-H, top-level port names come from txt objects on metal7
PORT LAYER POLYGON ports // this layer marks ports
PORT DEPTH PRIMARY
ATTACH ports metal7      // port polygons attached to metal7 polygons
                           // define ports
```

In Calibre nmLVS-H connectivity extraction, for example, you would see recognized ports in the run transcript, as follows:

```
-----
-----          PORTS
-----
(1.5,70.25) 50 TOPCELL
...
```

The coordinates of the port, its layer, and the cell in which the port appears are listed.

## Port Layer Text

Specification statement

### **PORT LAYER TEXT** *layer* [*layer* ...]

Used only in Calibre nmLVS/nmLVS-H and Calibre xRC.

#### Parameters

- *layer*

An original layer or layer set. You can specify *layer* any number of times in one statement.

#### Description

Causes Port Layer Text objects on the specified layer(s) to be read and treated as top-level ports in geometric databases. The port layer, location, and name are the same as the *layer*, location, and value of the text object, respectively.

Port text objects can be attached to port shapes using explicit attachment (see [Attach](#)), implicit attachment (both port text and port shapes are on the same [Connect](#) layer), or free attachment (see [Label Order](#)). Port Layer Text object names are netlisted during connectivity extraction as top-level ports.

Reading of port objects into the top-level cell is controlled by the [Port Depth](#) specification statement. Port objects from the specified depth are transformed to top-level space and replicated according to the hierarchical structure of the design. Port objects from lower levels of the hierarchy behave as if they were at the top level. This is true in both flat and hierarchical applications.

In hierarchical applications, Port Layer Text object names in the top-level cell are output by the hierarchical SPICE netlister (calibre -spice) as top-level subcircuit pin names. Port objects at lower levels of the hierarchy do not specify cell pin names. They can, however, be used by PEX applications.

Port text objects at any level of the hierarchy can overlap shapes at lower levels of the hierarchy and can be attached to those lower-level shapes. The hierarchical connectivity extractor forms any pins in the lower-level cells that are necessary to connect to port objects higher up. This is similar to how layout text is handled hierarchically.

For the top-level cell, the hierarchical SPICE netlister outputs port names in the pin list of the corresponding top-level subcircuit. The name chosen for the port is either the Port Layer Text object name or the user-given net name, whichever is present. If both are present, the net name prevails. If more than one Port Layer Text object is attached to a single net, and the net has no user-defined name, then one of the port object names is chosen arbitrarily.

#### Note



1. Text objects defined in the rule file (by [Text](#) specification statements) do not become ports.
2. Reading of text ports does not depend on [Text Layer](#) or [Text Depth](#) specification statements.

## Port Layer Text

---

In Calibre xRC this statement specifies the ports and xcell pin layers.

For additional information, see “Netlisting of Ports and Pins” in the *Calibre Verification User’s Manual*.

See also [Port Layer Polygon](#).

## Examples

Top-level port names are not assigned by default during nmLVS-H connectivity extraction. For example, assume you have the following in the rule file:

```
// Implicit attachment is used  
  
TEXT LAYER metal1 // for leaf cells  
TEXT LAYER metal2 // for routing cells  
CONNECT metal1 metal2 by via
```

Then the top level of the extracted netlist might appear as follows:

```
.SUBCKT TOPCELL           $$ no port names  
** N=4 EP=0 IP=8 FDC=6  
X0 GND 1 PWR CellA $T=0 0 0 0 $X=-4250 $Y=0  
X1 1 GND PWR CellB $T=29000 0 0 0 $X=24500 $Y=0  
.ENDS
```

You can add the following statements to the rule file:

```
LAYER port_text 50  
PORT LAYER TEXT port_text  
ATTACH port_text metal2  
  
PORT DEPTH PRIMARY
```

and label the metal2 ports as GND and PWR appropriately. Then the extracted netlist would appear as follows:

```
.SUBCKT TOPCELL GND PWR      $$ port names now appear  
** N=4 EP=2 IP=8 FDC=6  
X0 GND 1 PWR CellA $T=0 0 0 0 $X=-4250 $Y=0  
X1 1 GND PWR CellB $T=29000 0 0 0 $X=24500 $Y=0  
.ENDS
```

## Ports

Layer operation

### **PORTS *layer***

Used only in Pyxis Layout.

#### Parameters

- *layer*  
A required original layer or layer set.

#### Description

Generates a derived polygon layer consisting of all the port shapes on *layer*. Selects the port shapes only from the top-level cell of the active window.

#### Examples

```
x = PORTS metall1 // all ports on metall1
```

# Precision

Specification statement

**PRECISION { *number* | {*integer1 integer2*} }**

## Parameters

- ***number***  
A required, positive, floating-point value that specifies the database precision.
- ***integer1 integer2***  
Positive integers where the quotient *integer2* / *integer1* becomes the precision. These parameters must be specified if ***number*** is not specified.

## Description

Defines the ratio of database units to user units. A database unit (dbu) is the unit measure of length for the input database. This statement is optional and can appear once in a rule file. In Calibre, the default Precision is 1000. This is the reciprocal of the common layout database precision of 0.001.

All numeric values of length within a rule file are in user units. However, when a rule file is compiled, user units are converted into an equivalent number of database units by multiplying the user units by the number specified in the Precision statement. For example, if you assume a user unit of microns, and if your rule file Precision is 1000, this implies there are 1000 dbu per micron.

This statement affects the precision with which length measurements are taken. For example, Precision 1000 enables measurements to be taken to the nearest 0.001 user unit, whereas if your rule file Precision is 100, measurements are taken to the nearest 0.01 user unit. If a constraint in a layer operation requires a greater Precision than is currently specified in the rule file, then a NUM 8 compiler error is generated.

If your database precision does not match the rule file Precision, a compiler error results. You can override this error with the [Layout Input Exception Severity](#) PRECISION\_RULE\_FILE 1 setting, which allows the condition but issues a warning. If you choose to override the error, then the rule file Precision is used, and the data is scaled up or down according to the Precision you specify. For example, if your database precision is 0.001 and your rule file Precision is 10000, then layout polygon coordinates are scaled down by a factor of 10.

You can specify the value of the database precision separately by using the [Layout Precision](#) statement. The [Layout Path](#) and [Layout Path2](#) statements using the PREC keyword serve the same purpose. If you specify any of these statements, the values they contain are used to verify the input database precision rather than the Precision statement value. This method is used for verifying the database precision only; it does not change the measurement precision used during the run.

For most applications, specifying ***number*** as an integer value is desirable; non-integer values are rarely used. If A and B are integers, specifying Precision A B is equivalent to Precision (B/A). For example, Precision 10 1000 is equivalent to Precision 100.

The **number**, **integer1**, and **integer2** parameters may be names associated with numeric-valued environment variables. In this case, the value of the variable is resolved in the shell environment. If the variable cannot be resolved properly, the ENV1 or ENV2 compiler error is generated. Additionally, the values assigned to these parameters define the values for the system variables \$PRECISION, \$PRECISION\_N, and \$PRECISION\_D, respectively. See the [Variable](#) statement for more information on system variables and declaring environment variables.

The [Layout Polygon](#), [Polygon](#), [Layout Text](#), and [Text](#) specification statements do not have their objects' coordinates scaled up or down when you change your Precision setting. Coordinates specified in these statements are always interpreted in absolute user units. This can have unexpected results for some users. For example, if you have the following in your rule file:

```
PRECISION 1000
POLYGON 0 0 1 1 layer1
```

This places a square object with opposing corners at vertices (0, 0) (1, 1) on layer1, with the coordinates in microns. If you change the Precision to 2000 (effectively scaling the database down by half as it is read in) but leave the Polygon statement unchanged, this results in the square having corners at (0, 0) (1, 1), *not* (0, 0) (0.5, 0.5) as would be the case for a layout database polygon. This is so that the polygon has the stated dimensions of  $1 \times 1$  user units. Any further scaling of the square (such as with the [Layout Magnify](#) statement) is applied in the usual way.

By default, DRC results written to the nmDRC results database are written with the same Precision as what is specified in the rule file. You can change the precision of the results database by using the [DRC Results Database Precision](#) specification statement.

As indicated previously, any scaling performed by the Layout Magnify specification statement is in addition to any scaling that occurs due to a difference between your rule file Precision and your database precision.

If writing out mask data and magnifying the layout by a value that is less than 1 (such as for Layout Magnify or [DRC Magnify Results](#)), it is recommended that the output database precision be greater than the input physical precision by a factor of 10. This reduces the number of gaps in layers due to grid snapping. The output database precision follows the Precision statement setting by default.

In Pyxis Layout, the **number** parameter and the value of the \$precision Process variable must match; otherwise ICrules generates a compiler error. If you do not include the Precision statement in a rule file, the default database precision is equal to the value of the \$precision Process variable.

To check your user grid, see [Resolution](#).

See “[Layout Magnification](#)” in the *Calibre Solutions for Physical Verification* manual for a complete discussion of precision and magnification effects.

See also [Layout Use Database Precision](#) and [DRC Magnify Results](#).

## Examples

### Example 1

If your database precision is 0.001 and your user unit of length is microns ( $1 \times 10^{-6}$  meter), setting a rule file Precision of 1000 implies the database unit is nanometers ( $1 \times 10^{-9}$  meter).

```
PRECISION 1000
// 1000 database units to 1 user unit; dbu = nanometer
```

Changing the rule file Precision to 10000 implies a scaling down of the database by a factor of 10, and a measurement precision of 0.0001 microns. To compensate, you could magnify the layout by a factor of 10:

```
// Layout precision is 0.001
LAYOUT INPUT EXCEPTION SEVERITY PRECISION_RULE_FILE 1
// database precision and rule file PRECISION do not match
PRECISION 10000 // database scaled down by factor of 10
LAYOUT MAGNIFY 10 // keep data at original scale
```

Or you can do this:

```
PRECISION 10000 //scale down by factor of 10
LAYOUT PRECISION 1000 //specify the expected database precision
LAYOUT MAGNIFY AUTO //automatically magnify by factor of 1000/1000
```

Additional related examples are found under [Layout Magnify](#) and [DRC Magnify Results](#).

### Example 2

This example shows how to scale mask data by 50%:

```
LAYOUT PRECISION 1000
PRECISION 10000 // increase precision by a factor of 10
// to reduce grid snapping; data scaled by 1/10
LAYOUT MAGNIFY 5 // use a 0.5 magnification
```

The output data has the same physical precision as the Precision statement by default.

# Push

Layer operation

**PUSH** *layer* [LOCATE *layer2*] [LIGHT | MEDIUM]

## Parameters

- *layer*  
A required original layer or a derived polygon layer.
- LOCATE *layer2*  
An optional keyword and layer name that directs the operation to instantiate output geometry (that is, pushed geometry from *layer*) into cells that contain *layer2* geometry anywhere in their hierarchy.
- LIGHT  
An optional keyword that restricts the cells into which an object can be pushed to just those cells that are uniquely placed. May not be specified with MEDIUM.
- MEDIUM  
An optional keyword that restricts the cells into which an object can be pushed. This setting is not as restrictive as the LIGHT option, nor as permissive as the default behavior. May not be specified with LIGHT.

## Description

### Note



This operation is intended to change the hierarchical location of the input layer and write it to the output layer. The exact hierarchical location of the output is difficult to predict. This can have adverse impacts. Hence, this operation should generally be avoided.

For hierarchical Calibre applications, this operation creates a copy of the input *layer* where the geometry is pushed down the hierarchy to the lowest possible level. The lowest possible level is the lowest level in which instantiation of the output geometry remains identical in each placement of a cell.

For flat Calibre applications and ICverify, this operation is equivalent to the [Copy](#) operation.

This operation is useful in hierarchical applications for improving the hierarchical instantiation of layers with *large shapes that cover vast numbers of flat placements*, such as a memory array.

The LOCATE keyword is useful in directing the destination of pushed geometry to only those cells having *layer2* geometry. For example:

```
x = LITHO poly FILE setup.litho
/* Push x back down the hierarchy, if possible, but not haphazardly. Try
to keep it where poly was to begin with. */
y = PUSH x LOCATE poly
```

## Push

---

```
poly_out { copy y }
DRC CHECK MAP poly_out GDSII 10 poly_out.gds
```

The [Inside Cell](#) operation can be useful in isolating the hierarchical level of the LOCATE layer.

In certain cases in LVS device recognition, seed shapes can be derived from layers at multiple levels of hierarchy, including the top level. This causes devices to be recognized at undesired levels of hierarchy. The best solution is not to derive the seed shapes in the manner described previously. Rather, the device layer at the top level should be declared as a [Device](#) auxiliary layer and left out of the derivation of the device seed shape. If that solution is not possible, then the Push LOCATE option can be useful in pushing seed shapes into the cell of origin. The LOCATE *layer* should be a seed-forming layer in the cell of origin.

The Push operation can be slow in certain situations, especially when the input geometry is so random that low-level template-specific instantiation is minimal. The LIGHT keyword specifies that the objects can only be pushed into cells that are uniquely placed from the hierarchical viewpoint. The LIGHT option is much faster in these situations and is beneficial in preserving scalability in parallel Calibre runs by redistributing heavily flattened geometry to Calibre “bin cells” (ICV\_ cells), for example.

The MEDIUM keyword offers a compromise between the LIGHT behavior and the default behavior. Objects are pushed into cells other than uniquely-placed ones, but the pushing behavior is less than the default. The MEDIUM option is slower than LIGHT, but can be much faster than the default.

See also [Flatten](#) and [Push Cell](#).

## Examples

The following using the LOCATE option together with Inside Cell to isolate the level to which polygons are pushed:

```
push_target = INSIDE CELL layer1 cellA // this identifies the destination
pushed_layer = PUSH layer2 LOCATE push_target // push to the destination
```

# Push Cell

Specification statement

**PUSH CELL** *cell\_name* [*cell\_name* ...]

Used only in hierarchical Calibre.

## Parameters

- *cell\_name*

A required name of a cell. You can specify *cell\_name* any number of times in one statement. The *cell\_name* can be a string variable. The cell name parameters can contain one or more asterisk (\*) wildcard characters, where the \* matches zero or more characters. When using the \* character, enclose the cell name in quotation marks (" "); otherwise, a compilation error occurs because the asterisk is a reserved symbol.

## Description

Specifies cell names whose placements are to be completely flattened, then pushed to the lowest possible level by hierarchical Calibre applications. This statement may be specified any number of times.

This specification statement is useful (although design-specific) in pushing notch and gap fill cells (cells containing only notch or gap fill geometry) down the hierarchy. It can provide huge performance gains if the notch or gap fill geometry in the specified cells would be merged into (on a template-specific basis) geometry at a different hierarchical level.

See also [Flatten Cell](#), [Flatten Inside Cell](#), and [Push](#).

## Examples

```
PUSH CELL cella cellb // flatten these cells and push them to  
// the lowest possible level of the hierarchy
```

## Rectangle

Layer operation

**RECTANGLE** *layer* [*constraint1* [BY *constraint2*]] [ASPECT *constraint3*]  
 [ORTHOGONAL ONLY | MEASURE EXTENTS]

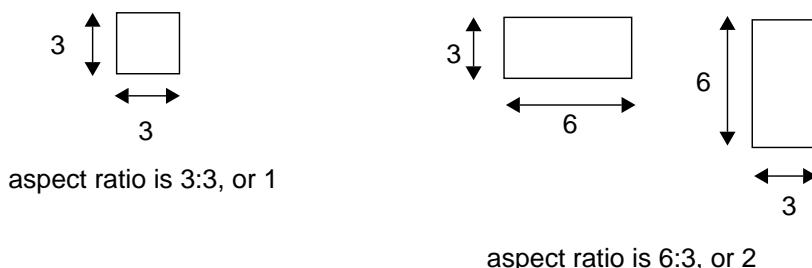
### Summary

Layer operation that selects rectangles based on geometric properties.

### Parameters

- *layer*  
An original layer or layer set, or a derived polygon layer.
- *constraint1*  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. A constraint must contain non-negative real numbers and is interpreted in user units.
- BY *constraint2*  
An optional keyword set, where *constraint2* must follow the same guidelines as *constraint1*. The optional keyword BY must always precede *constraint2*.
- ASPECT *constraint3*  
An optional keyword and constraint that specifies the ratio of the longer side to the shorter side that a rectangle must have in order to be output. [Figure 4-310](#) shows the aspect concept:

**Figure 4-310. ASPECT**



- ORTHOGONAL ONLY

An optional keyword which limits selection to rectangles having sides that are parallel (respectively) to the coordinate axes of the database. May not be specified with MEASURE EXTENTS.

- MEASURE EXTENTS

An optional keyword that selects (not necessarily rectangular) polygons from the input layer having individual rectangular extents that meet the constraint. Polygons that fit into the extents region defined by the constraint are output. May not be specified with ORTHOGONAL ONLY.

## Description

Selects all rectangles on *layer* if no constraint is specified. Can also select all *layer* rectangles with edge lengths conforming to the constraints. If you specify only *constraint1*, the rectangles that are selected have at least one pair of edges that satisfy the constraint. If you specify *constraint1* BY *constraint2*, the rectangles that are selected have one pair of edges that meets the first constraint and the other pair of edges meets the second constraint.

In the event that non-orthogonal (with respect to the coordinate axes) rectangles exist on *layer*, you should be careful when specifying constraints in the Rectangle operation. A potential problem arises when you make assumptions about the exact size of polygons. For example, you might want to avoid using constraints such as == 5 when *layer* contains non-orthogonal rectangles. A better choice of constraint is > 4.995 < 5.005. This allows a tolerance of ten database units, if your [Precision](#) is 1000.

If you specify ORTHOGONAL ONLY, this outputs rectangles where the dimensions satisfy the given constraints, if any, or the rectangles' edges are parallel to the database coordinate axes, respectively.

The MEASURE EXTENTS keyword is especially useful when attempting to select non-rectangular polygons, or rectangular polygons that are not orthogonal to the database axes, where the polygons fit within a specific rectangular extent. The dimensions of the rectangular extent are specified with a constraint.

See also [Not Rectangle](#) and [Enclose Rectangle](#).

## Examples

### Example 1

```
// Generate a layer containing all contacts that are rectangular.
rectangular_contacts = RECTANGLE contact
```

### Example 2

Using ASPECT.

```
/* Select contact rectangles having a side longer than 2 and an aspect
ratio greater than or equal to 3. */

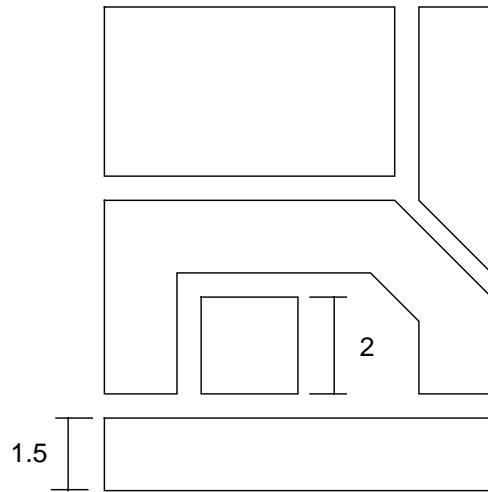
rule { RECTANGLE contact > 2 ASPECT >= 3 }
```

## Rectangle

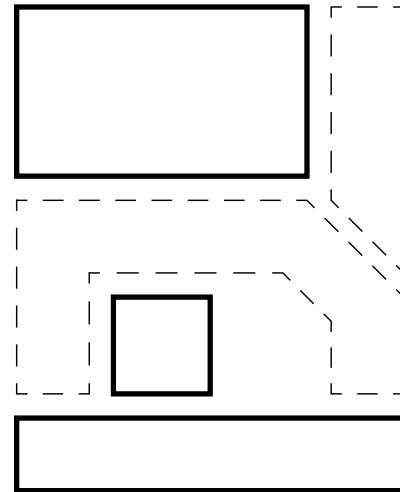
### Example 3

Figure 4-311 shows three Rectangle operations. The operation shown at top right selects all layer1 rectangles. The operation shown at bottom left selects only the layer1 rectangles that have a side longer than 2 user units. The operation shown at bottom right selects only the layer1 rectangles that have both sides longer than 2 user units.

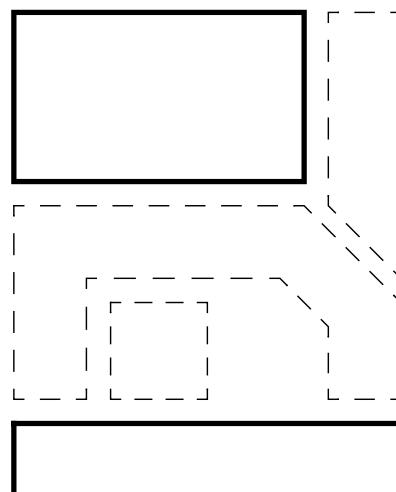
**Figure 4-311. Rectangle**



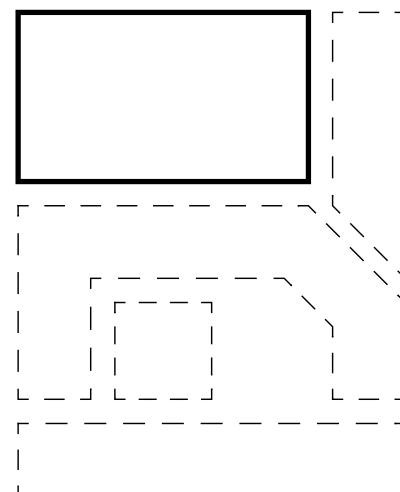
layer1 polygons



rectangle layer1



rectangle layer1 > 2



rectangle layer1 > 2 by > 2

# Rectangle Enclosure

Layer operation

**RECTANGLE ENCLOSURE** *layer1 layer2 [intersection\_filter]*  
 [OUTSIDE ALSO] [ORTHOGONAL ONLY] *rectangle\_rule* [*rectangle\_rule* ...]

## Summary

Provides efficient checking of multiple-condition enclosure rules for rectangular polygons.

## Parameters

- ***layer1***  
 An original layer or a derived polygon layer.
- ***layer2***  
 An original layer or a derived polygon layer.
- ***intersection\_filter***  
 An optional parameter group that permits intersecting edge pairs to be measured. The value of *intersection\_filter* takes the following form:

ABUT [*constraint*] [SINGULAR]

You may specify any combination of the optional keywords ABUT and SINGULAR in one operation:

ABUT [*constraint*] — Specifies to measure the separation between intersecting edges in addition to non-intersecting edges (the default). Intersecting edges are measured only if the appropriate angle between them conforms to the optional constraint (interpreted in degrees).

The optional *constraint* must contain non-negative real numbers less than 180. Single-operator constraints such as < 90 and > 135 are interpreted as  $\geq 0 < 90$  and  $> 135 < 180$ . The default value if you do not specify a constraint is  $\geq 0 < 180$ .

If the *constraint* modifier includes zero in its range (for example, < 90, == 0,  $\geq 0 < 45$ ), then any edges A of X and B of Y that are *coincident inside* (see [Coincident Inside Edge](#) for a description), are also output. This is because the angle between the exterior side of A and the interior side of B is zero. There is no measurement involved in this event, and polygon containment criteria are not applied.

SINGULAR — Instructs dimensional check operations to measure the separation between the corresponding sides of intersecting edges at points of polygon singularity. Singularities are point-to-edge or point-to-point polygon intersections or self-intersections, and are normally design rule errors. Figure 4-89 shows examples of singularities detected by an Enclosure operation.

When a dimensional check operation detects a point of singularity, all edges forming the point of singularity are measured as if an unconstrained ABUT parameter were specified in the operation. This overrides any specified ABUT parameter, but only at the point of singularity.

- OUTSIDE ALSO

Optional keyword that outputs rectangles from *layer1* that are not enclosed by *layer2*. When you specify OUTSIDE ALSO, rectangles from *layer1* outside or coincident outside with *layer2* are output by the operation (Figure 4-90). See [Outside Edge](#) or [Coincident Outside Edge](#) for explanations of related ideas.

- ORTHOGONAL ONLY

Optional keyword that specifies only rectangles with edges parallel (respectively) to the database coordinate axes are processed by the Rectangle Enclosure operation.

- *rectangle\_rule*

This parameter specifies how *layer1* rectangles enclosed by *layer2* are to be measured. At least one *rectangle\_rule* must be provided and has the following form:

**GOOD | BAD** {*value1* [metric] *value2* [metric] *value3* [metric] *value4* [metric]}

You must specify either **GOOD** or **BAD** followed by the four *value* arguments shown. The *metric* argument is optional.

**GOOD | BAD** — Controls the type of *rectangle\_rule* you are specifying. GOOD-type rectangle rules do not output rectangles that satisfy them. BAD-type rectangle rules output rectangles that satisfy them. Under typical circumstances, there should be at least one GOOD rule in a rule check. This is discussed in detail later in this section.

*value1* ... *value4* — Non-negative, floating-point numbers used to check edge extensions of enclosed rectangles. Enclosure spacing is measured as < *valueN*. Spacing < 0 is not considered. Values of 0 are incompatible with the OPPOSITE or OPPOSITE EXTENDED metric.

*metric* — Controls the measurement metric used within the *rectangle\_rule*. (Euclidean is the default). This parameter must be used in conjunction with a *value1* ... *value4* argument. OPPOSITE, OPPOSITE EXTENDED *value*, and SQUARE are the other possible metrics. The value 0 may not be specified with OPPOSITE or OPPOSITE EXTENDED. See “Metrics” in the [Calibre Verification User’s Manual](#) for a complete discussion of metrics.

## Description

Rectangle Enclosure is used for efficient enclosure checking of rectangles when multiple rules may apply to the enclosed rectangles. Complex enclosure rules involving rectangular polygons are handled more efficiently by this operation than by multiple applications of the [Enclosure](#) operation. An example of such an enclosure rule follows:

Contacts must be enclosed by metal by 0.15. However, two opposite sides can each be as close as 0 if the other two sides are at least 0.5. Two opposite sides can each be as close as 0.05 if the other two sides are at least 0.4. Two opposite sides can be as close as 0.1 if the other two sides are at least 0.3. Finally, if all sides are at least 0.15, then the contact is OK.

You can check this rule as follows (assuming the contacts must be fully enclosed, and that singularities and acute abutments are errors):

**RECTANGLE ENCLOSURE contact metal ABUT > 0 < 90 SINGULAR OUTSIDE ALSO**

**GOOD 0.00 0.50 0.00 0.50**

**GOOD 0.05 0.40 0.05 0.40**

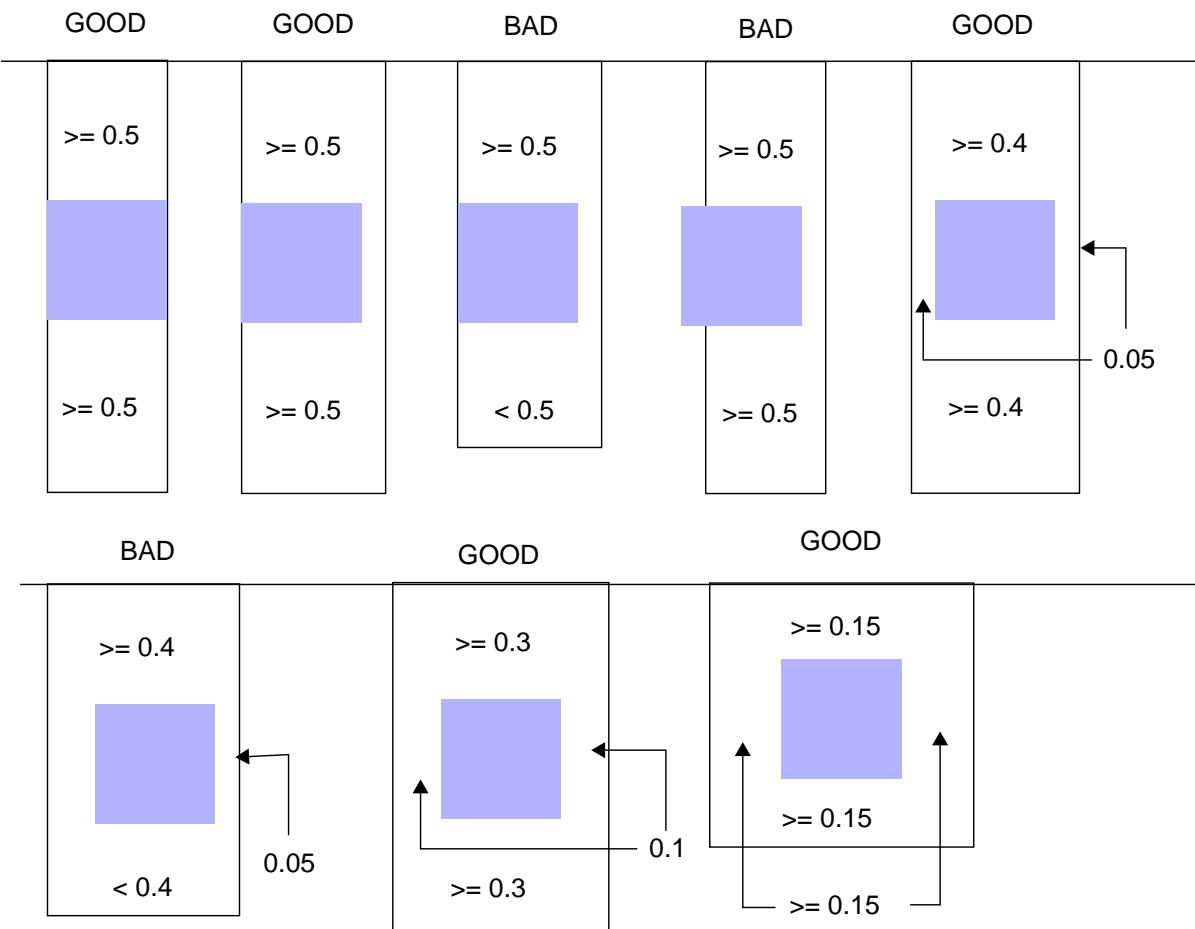
**GOOD 0.10 0.30 0.10 0.30**

**GOOD 0.15 0.15 0.15 0.15**

The tool checks each enclosure of contact by metal against the four spacing conditions listed *in the order the conditions appear*. Any rectangle that does not satisfy at least one of the four conditions is classified as BAD and is output. Note that you do not need to explicitly specify any BAD conditions in order for a contact to be classified as BAD.

Figure 4-312 shows some examples of possible results from this check:

**Figure 4-312. Rectangle Enclosure Check**



The basic algorithm used for Rectangle Enclosure follows:

1. For each polygon R in *layer1* (contact in the above rule check), do the following:
  - If R is not a rectangle, then it is output and the algorithm continues to the next polygon in *layer1*.
  - If ORTHOGONAL ONLY is specified and R is not orthogonal to the database axes, the algorithm continues to the next polygon in *layer1*.
  - If any of the global parameters ABUT, SINGULAR, or OUTSIDE ALSO are specified, then R is measured according to this nmDRC operation, with the appropriate keywords applied:

ENCLOSURE R Y < delta [ABUT [*constraint*]] [SINGULAR] [OUTSIDE ALSO]

The delta is a small positive value (Y corresponds to metal in the rule check shown previously). If there is any output due to the secondary keywords, then R itself is output and the operation continues to the next polygon in *layer1*. Otherwise, R is tested as in step 4.

4. Each individual *rectangle\_rule* from first to last is then examined. If R satisfies the *rectangle\_rule*, then processing of R is complete. If it is a BAD-type *rectangle\_rule*, then R is output. If it is a GOOD-type *rectangle\_rule*, then R is not output. If R does not satisfy a *rectangle\_rule* then it is output (that is, it is considered BAD).

(The implication of this step is there must be at least one GOOD *rectangle\_rule* in a given rule check, or all R will be output as bad.)

To determine if a rectangle R satisfies a *rectangle\_rule*, an arbitrary edge on R is chosen and named A. The next edges, in clockwise order, are named B, C, and D. Eight relations are then constructed, each with four (edge,*value* [OPPOSITE]) components. For example:

1. {(A,value1), (B,value2), (C,value3), (D,value4)}
2. {(A,value4), (B,value1), (C,value2), (D,value3)}
3. {(A,value3), (B,value4), (C,value1), (D,value2)}
4. {(A,value2), (B,value3), (C,value4), (D,value1)}
5. {(A,value1), (B,value4), (C,value3), (D,value2)}
6. {(A,value4), (B,value3), (C,value2), (D,value1)}
7. {(A,value3), (B,value2), (C,value1), (D,value4)}
8. {(A,value2), (B,value1), (C,value4), (D,value3)}

If any of these relations satisfies the *rectangle\_rule* then R itself is said to satisfy the *rectangle\_rule* and processing of R is complete. Note that you may specify the OPPOSITE metric with any *value* argument and it would be taken into account in each of these relations as a (edge,*value* OPPOSITE) component.

An orthogonal rectangle R may be defined on *layer1* to satisfy a GOOD-type *rectangle\_rule* prior to invoking the normal edge measurement procedures. If R is inside of *layer2*, then the *rectangle\_rule* is satisfied and no further processing of R is required. Using this feature minimizes the possibility of introducing errors when adding geometry on the second input layer, eliminates flat and hierarchical discrepancies, and improves performance. This situation applies only when all *rectangle\_rules* are of the GOOD-type. Refer to [Example 2](#).

A relation satisfies a GOOD-type *rectangle\_rule* if all four components of that relation are GOOD. A relation satisfies a BAD-type *rectangle\_rule* if all four components are BAD. Determination of whether a specific component (edge,*value* [OPPOSITE]) is GOOD or BAD proceeds as follows:

1. Let I = edge NOT OUTSIDE EDGE Y
2. Measure I to Y according to the nmDRC operation

**ENCLOSURE I Y < *value* ABUT == 0 [OPPOSITE]**

If the default Euclidean metric is used, then the PERPENDICULAR ALSO option (as in [Enclosure](#), [External](#), and [Internal](#) operations) is used.

3. The component is GOOD if there is no output, else it is BAD.

If the *value* is 0, then the component is defined to be GOOD in a GOOD-type *rectangle\_rule* and BAD in a BAD-type *rectangle\_rule*.

Using all BAD conditions in a rule check results in *all* rectangles on *layer1* being output as BAD because a rectangle can either satisfy a BAD condition, or no condition; in either case, the rectangle is output. This is normally not useful. Therefore, there should be at least one GOOD condition a rectangle can satisfy in a rule check. For most applications, it is sufficient to specify all of the possible GOOD rectangle configurations and let the tool decide which polygons are BAD.

If you have only a few BAD configurations to check, specify them first in your rule check followed by a fall-through GOOD condition. For example:

**RECTANGLE ENCLOSURE contact metal ABUT > 0 < 90 SINGULAR**

**BAD 0.1 0.1 0.1 0.1**

**GOOD 0 0 0 0**

In this example, you are only interested in finding contacts that have less than 0.1 user units of enclosing metal around them. Whatever contacts satisfy the BAD condition are output, everything else is GOOD. You would not want to list the GOOD condition first in this example, because all contacts would be classified as GOOD.

## Rectangle Enclosure

---

Returning to the enclosure rule at the beginning of this section, if opposite side were replaced by adjacent side, the following applies:

**RECTANGLE ENCLOSURE contact metal ABUT > 0 < 90 SINGULAR OUTSIDE ALSO**

**GOOD 0.00 0.50 0.00 0.50**  
**GOOD 0.05 0.40 0.05 0.40**  
**GOOD 0.10 0.30 0.10 0.30**  
**GOOD 0.15 0.15 0.15 0.15**

If opposite side were replaced by side, all possible cases must be considered:

**RECTANGLE ENCLOSURE contact metal ABUT > 0 < 90 SINGULAR OUTSIDE ALSO**

**GOOD 0.00 0.50 0.00 0.50**  
**GOOD 0.05 0.40 0.05 0.40**  
**GOOD 0.10 0.30 0.10 0.30**  
**GOOD 0.15 0.15 0.15 0.15**  
**GOOD 0.00 0.00 0.50 0.50**  
**GOOD 0.05 0.05 0.40 0.40**  
**GOOD 0.10 0.10 0.30 0.30**  
**GOOD 0.15 0.15 0.15 0.15**

The default behavior of Rectangle Enclosure dictates that non-rectangles are not output by the operation without any further processing. The ORTHOGONAL ONLY keyword causes any polygon that is not an orthogonal rectangle to be output without further processing.

## Examples

### Example 1

Consider the rule:

Contacts must be enclosed by metal1 by at least 0.005. However, any contact corner must be enclosed by metal at 0.06; that is, no two portions of contact edges enclosed at less than 0.06, in an OPPOSITE metric sense, can intersect.

This rule can be checked as follows:

```
rule {
    X = RECTANGLE ENCLOSURE contact metal1 ABUT < 90 SINGULAR
    OUTSIDE ALSO
    GOOD 0.005 OPPOSITE .06 OPPOSITE 0.005 OPPOSITE .06 OPPOSITE
    ENC X metal1 < 0.005 ABUT < 90 SINGULAR OVERLAP OUTSIDE ALSO
    Y = ENC [X] metal1 < .06 OPPOSITE
    INT Y < 0.01 ABUT == 90 INTERSECTING ONLY
    // 0.01 is arbitrary.
}
```

The Rectangle Enclosure operation is used here for efficiency. In most designs, errors are the exception rather than the norm and you expect that almost all contacts would satisfy the GOOD

portion of the Rectangle Enclosure operation. Hence, X would be very small, or empty. The remainder of the rule would be very fast.

If Rectangle Enclosure were not used and, assuming most contacts were constructed as close as possible to metal1, then

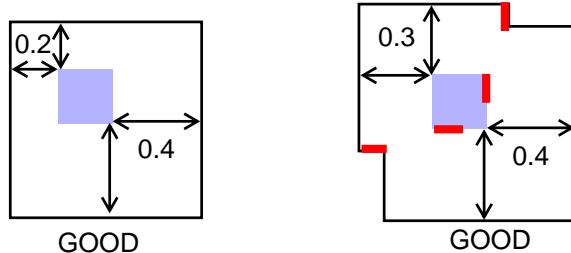
#### **ENC [X] metal1 < 0.06 OPPOSITE**

would generate vast numbers of edges in layer Y and consume a great deal of time and memory. It would be much slower than if most contacts were removed with Rectangle Enclosure first.

#### **Example 2**

Consider the following situation:

□ X  
■ Y



```
RECTANGLE ENCLOSURE Y X
GOOD 0.2 0.2
0.4 OPPOSITE EXTENDED 0.4
0.4 OPPOSITE EXTENDED 0.4
```

The figure on the right has additional layer X area surrounding layer Y, so it is intuitive that both figures are classified as GOOD.

However, in the figure on the right, there are edge pairs (marked in red) that fall within the OPPOSITE EXTENDED measurement region around the layer Y shape; so, one might expect an error. However, such a result would contradict the intent of the rule. If the orientation on the left is good, certainly the orientation on the right should be also.

To allow the right figure to pass the check, an additional measurement rule is added internally by the tool to the existing Rectangle Enclosure operation, which ensures that the polygon orientation on the right passes the check.

This measurement rule “expands” R by the dimensions specified in the operation. There are eight possible configurations for such an expanded rectangle. If any of the eight configurations fit inside the layer Y polygon, then R is considered GOOD and the OPPOSITE EXTENDED metric is not applied.

# Rectangles

Layer operation

**RECTANGLES** *width length {spacing | {width\_spacing length\_spacing}}*  
 [OFFSET {*offset* | {*width\_offset length\_offset*}}]  
 [{INSIDE OF *x1 y1 x2 y2*} | {INSIDE OF LAYER *layer*}]  
 [MAINTAIN SPACING]

## Summary

Creates an array of rectangles of specified dimensions and spacing. Often used in planarization and layer fill applications.

## Parameters

- ***width length***

A required pair of positive, floating-point numbers that indicate the width (x-axis) and length (y-axis) of a rectangle, in user units.

- ***spacing***

A required, positive, floating-point number that indicates the spacing in user units, in both the x- and y- directions, between rectangles. When you use this option, the ***width\_spacing length\_spacing*** option cannot be used.

- ***width\_spacing length\_spacing***

A pair of required, positive, floating-point numbers that indicate the width spacing (x-axis) and length spacing (y-axis), respectively, between rectangles. When you use this option, the ***spacing*** option cannot be used.

- **OFFSET {*offset* | {*width\_offset length\_offset*}}**

An optional keyword that specifies the horizontal and vertical offsets between adjacent rectangles. When you use this keyword, it must be followed by either the ***offset*** option or the ***width\_offset length\_offset*** option.

***offset*** — A positive, floating-point number that specifies both the x-axis and y-axis offsets, in user units, between rectangles. This value may be greater than the ***spacing*** value, in which case, the minimum spacing between rectangles is controlled by the ***spacing*** value. When you use this option, the ***width\_offset length\_offset*** option cannot be used.

***width\_offset length\_offset*** — A pair of positive, floating-point numbers in user units that indicate the x-axis and y-axis offsets, respectively, between rectangles. These values may exceed the respective ***width\_spacing*** and ***length\_spacing*** values, in which case the minimum spacing between rectangles is controlled by the ***spacing*** values. When you use this option, the ***offset*** option cannot be used.

- **INSIDE OF *x1 y1 x2 y2***

An optional keyword set that specifies an area within the specified extent to be filled with rectangles. The keyword must be followed by a set of four floating-point numbers in user

units, which indicate the lower-left ( $x_1, y_1$ ) and upper-right ( $x_2, y_2$ ) corners of the extent to be filled. If a coordinate is negative, it must be enclosed in parentheses ( ).

- INSIDE OF LAYER *layer*

An optional keyword set that specifies a *layer* having polygons to be filled with rectangles. The *layer* indicates the name of an original or derived polygon layer.

- MAINTAIN SPACING

An optional keyword that controls the spacing of rectangles, such that a halo area is constructed around each rectangle, whereby no other rectangle may fall within the *spacing*, or *width\_spacing* and *length\_spacing* parameters.

## Description

Generates an output layer consisting of an array of rectangles with the specified dimensions and spacing. This layer extends to the portion of the database extent actually read in at run time and does not preserve nets. (See “[Extent](#)” for a discussion of database read-in.) You are able to control the size, spacing, offset, and extent of the generated layer using the appropriate parameters.

## Flat Applications

The following describes the semantics of the Rectangles operation in flat Calibre and ICverify applications.

Rectangles are aligned with the width along the x-axis and the length along the y-axis. Exact spacing of the rectangles along the width and length axes are given by the input parameters: *width*, *length*, and either *spacing*, or *width\_spacing* and *length\_spacing*.

The placement of the first rectangle always begins at the lower left corner of the database extent. If the rule file contains a [Resolution](#) value, the lower-left corner of the first rectangle is snapped (up and right) by that value. This ensures that the rectangles are placed on grid if the width, length, spacing, and offset parameters are appropriately specified.

The default algorithm is used if each optional offset value is less than or equal to the corresponding spacing value and MAINTAIN SPACING is not specified. The algorithm constructs columns of rectangles beginning in the lower-left corner of the area to be filled. In each column, a rectangle always begins at the point ( $X + AX$ ,  $Y + AY$ ), where:

- **X** is the lower-left x-coordinate of the previous rectangle.
- **AX** is the *width\_offset* (or *offset*) value.
- **Y** is the lower-left y-coordinate of the previous rectangle.
- **AY** is the *length* value plus the *length\_spacing* (or *spacing*) value.

This process continues until no further rectangles can be placed within the extent.

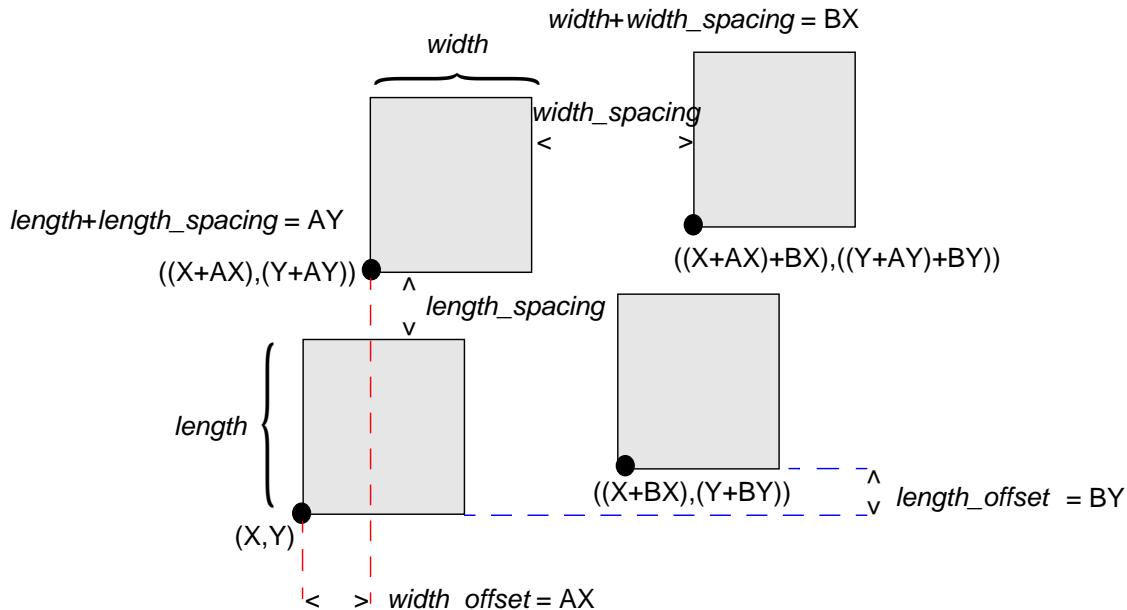
When moving left-to-right, a new column begins at the point ( $X + BX$ ,  $Y + BY$ ) where:

- **X** is the lower-left x-coordinate of the first rectangle in the previous column.

- **BX** is the **width** value plus the **width\_spacing** (or **spacing**) value.
- **Y** is the lower-left y-coordinate of the first rectangle in the previous column.
- **BY** is the **length\_offset** (or **offset**) value.

This process continues until no further rectangles can be placed within the extent (see Figure 4-313).

**Figure 4-313. Rectangles Parameters**



When using the offset options, a starting point can occur where a full rectangle (including any spacing requirements) would fit below it. In this case, the column is started at  $(X + CX, Y - CY)$  where:

- **X** is the lower-left x-coordinate of the first rectangle in the previous column.
- **CX** is the **width** value plus the **width\_spacing** (or **spacing**) value minus the **width\_offset** (or **offset**) value.
- **Y** is the lower-left y-coordinate of the first rectangle in the previous column.
- **CY** is the **length** value plus the **length\_offset** (or **offset**) value.

These left-to-right algorithms ensure a constant corner-to-corner separation of  $(DX, DY)$  between columns where:

- **DX** is the **width\_spacing** (or **spacing**) value minus the **width\_offset** (or **offset**) value.
- **DY** is the **length\_spacing** (or **spacing**) value minus the **length\_offset** (or **offset**) value.

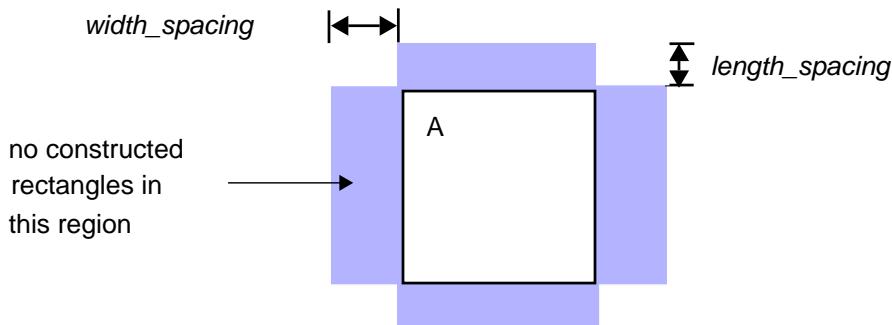
This corner-to-corner separation is determined by the distance between the second rectangle in the first column and the first rectangle in the second column, when moving from left-to-right.

When moving from bottom-to-top (filling the upper-left region of the database extent), the process is similar to that of the left-to-right algorithms described above, with appropriate modifications of the (X, Y) parameter order.

Corner-to-corner spacing can be as low as zero, when an offset value equals its corresponding spacing value.

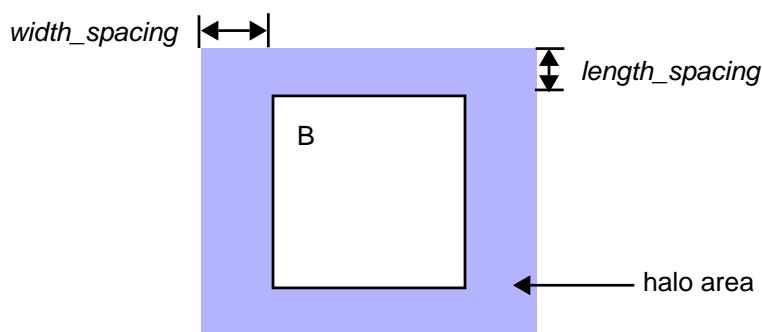
Given a constructed rectangle A, Figure 4-314 shows the region in which another constructed rectangle may not be placed:

**Figure 4-314. Rectangles With Default Spacing**



If you specify a *length\_offset* or *width\_offset* (or *offset*) that exceeds the corresponding spacing value, or if you specify the MAINTAIN SPACING optional keyword, then Rectangles spacing is controlled so that, given any constructed rectangle B, no other constructed rectangle is strictly within a halo area around B defined by the *width\_spacing* and *length\_spacing* (or *spacing*) as shown Figure 4-315:

**Figure 4-315. Rectangles With Maintained Spacing**



This spacing is controlled in the algorithm, along with allowing offset values to exceed their corresponding spacing values, by ensuring that the first rectangle in each new column obeys the *width\_spacing* to all rectangles in the previous column. This occurs within the y-extent of that first rectangle plus the *length\_spacing* in both directions.

All rectangles must fit entirely within the extent in order to be output.

If you specify INSIDE OF, then the specified area substitutes for the database extent. This area must be a valid, orthogonal rectangle.

If you specify INSIDE OF LAYER, then the rectangles are placed inside of the input layer polygons, according to the following:

1. For each polygon on the layer, compute its rectangular extent.
2. Fill the rectangular extent of the polygon according to the general steps discussed above.
3. Output complete rectangles that are inside the polygon boundary.

## Hierarchical Applications

In hierarchical applications, the algorithm used is more complicated.

As in the flat algorithm, the placement of the first rectangle always begins at the lower left corner of the database extent. If the rule file contains a [Resolution](#) value, the lower-left corner of the first rectangle is snapped (up and right) by that value. This ensures that the rectangles are placed on grid if the width, length, spacing, and offset parameters are appropriately specified.

If you do not specify INSIDE OF LAYER, then the output rectangles are instantiated hierarchically as much as possible.

Cells in which rectangle instantiation occurs are selected to achieve an optimal hierarchical distribution. In each cell where rectangle instantiation occurs, a fill area is determined. The algorithm selects the fill area to ensure no minimum spacing violation between rectangles in different cell placements at any point in the hierarchy. The spacing value is the larger of the specified width-axis and length-axis spacing values. The algorithm then distributes the rectangles in the fill area with the width-axis along the x-axis of the cell extent.

The hierarchical fill algorithm described above attempts to achieve a hierarchical distribution at the expense of:

**Density** — In general, the rectangles may not be as densely packed (from the flat point of view) as in the flat applications, especially when positive offsets are specified. This is due to a need to observe minimum spacing of rectangles (as described above) between any two cell placements at any point in the hierarchy. Loss of density is normally less than 5%.

**Uniform rotation** — If the sizes along the width and length axes are different, rotation of placements can cause blocks of rectangles in the flat view to be aligned with the width along the x-axis. Other blocks are aligned with the width along the y-axis.

**Uniform alignment** — Hierarchical rectangle placement can generate multiple fill areas in each cell, which are filled separately (such as the use of the INSIDE OF LAYER option in flat applications). This can cause blocks of rectangles in a cell to be positively offset from each other. This would not be seen in flat applications.

If you specify INSIDE OF LAYER, then the *layer* parameter is converted to fully-merged form, and in each cell where the fully-merged form exists, the operation follows the flat application's algorithm for INSIDE OF LAYER with the width-axis along the x-axis of the geometry extent in cell space. A layer is converted to fully-merged form in hierarchical applications by promoting geometry up the hierarchy until only full polygons (in the flat view) are present in a

cell; that is, it is not possible for a polygon in a cell to grow due to any hierarchical interaction of the cell's placements.

However there is an important modification to the previously-stated placement rule. This modification insures transform-invariance. This means that the specified width, height, spacing, and so forth of the generated rectangles must also be aligned on the x-axis and y-axis, respectively, from the point of view of the top-level cell coordinate space.

Transform-invariance is accomplished by *not placing* rectangles (and adjusting upward the hierarchical position of the INSIDE OF LAYER geometry) with different width and height dimensions, spacings, or offsets in cells with multiple placements having inconsistent rotations from the flat point of view or, for cells with consistent 90- or 270-degree rotations from the flat point of view, swapping the rectangle width and height dimensions, spacings, and offsets accordingly.

If the *layer* parameter of INSIDE OF LAYER is flat (for example, output of a [Density](#) operation) or is in fully-merged form only at the top-level cell (for example, output of an [Extent](#) operation), then the output rectangles are instantiated at the top-level. This overrides the hierarchical distribution described previously. The appearance of the output rectangles in these cases will be more akin to their appearance in flat applications.

For example:

```
// Create 2x4 fill patterns at 2 user unit spacing in the inverse of the
// active layer. Observe spacing of 2 user units to active and poly and
// 3 user units to nwell edges.
X1 = SIZE ACTIVE BY 2 // For spacing of 2 to active.
X2 = SIZE POLY BY 2 // For spacing of 2 to poly.
X3 = ( SIZE NWELL BY 3 ) NOT ( SIZE NWELL BY -3 )
// For spacing of 3 to nwell edges.
X4 = ( EXTENT ) NOT ( X1 OR ( X2 OR X3 ) ) // Area to fill.
X5 = RECTANGLES 2 4 2 INSIDE OF LAYER X4 // Fill pattern.
```

## Examples

### Example 1

```
// generate grid of rectangles with width 3, length 4, and
// spacing of 1 unit in x- and y-direction.
fill1 = RECTANGLES 3 4 1
```

### Example 2

```
// generate rectangles with width 2, length 4, x-spacing 2,
// y-spacing 3, and an offset of 2 units in the x- and y-direction.
fill2 = RECTANGLES 2 4 2 3 OFFSET 2
```

### Example 3

```
// generate rectangles with width 4, length 6, spacing of 3 units in
// x- and y-directions, x-offset 1, y-offset 2, inside the poly layer.
fill3 = RECTANGLES 4 6 3 OFFSET 1 2 INSIDE OF LAYER poly
```

### Example 4

```
// generate rectangles with width 2, length 5, spacing of 2
// units in x- and y-direction, offset 1 in x- and
// y-directions, inside of rectangle with corners (20, 20) and (200, 200).
fill4 = RECTANGLES 2 5 2 OFFSET 1 INSIDE OF 20 20 200 200
```

# Resistance Connection

Parasitic extraction

## RESISTANCE CONNECTION *layer1 layer2 [n1 n2]*

### Parameters

- ***layer1***  
A required original layer or a derived polygon layer.
- ***layer2***  
A original layer or a derived polygon layer.
- **[*n1 n2*]**  
A pair of required strings that must be enclosed in brackets ( [ ] ).

The strings are described as follows:

***n1*** — A non-negative real number, which is resistance multiplied by area (ohms \* microns squared), that specifies the inverse area proportionality constant.

***n2*** — A non-negative real number, which is in units of resistance, that specifies the inverse edge proportionality constant. Typically, *n2* is 0.

For more information about units of resistance, refer to the [Unit Resistance](#) statement.

### Description

Defines the proportionality constants for computing the resistance of current flowing between two conduction layers. The resistance is based on the area and edge effects of the connection between *layer1* and *layer2* in the presence of a contact layer or via.

Input layers must appear in a [Connect](#) BY statement for the given connectivity mode and in the same layer order. The resistance value is based on the area and edge effects of the connection common to both *layer1* and *layer2*. Either or both of these effects can be modeled by specifying a non-zero number value.

You can define the *layer1* and *layer2* proportionality constants (*n1* and *n2*) with the Resistance Connection operation only once and in the given order of *layer1* and *layer2*.

The extractor uses this equation to compute the connection resistance:

$$R = \frac{n1}{\text{common contact area}} + \frac{n2}{\text{common contact perimeter}}$$

In resistance fracturing, there are two different dimensions, the width and the length.

If you set *n2*=0, this is equivalent to long wires being limited to 100um in length, but two-dimensional fracturing is not performed (no width fracturing). This is good for wide interconnect wires.

## Resistance Connection

If you set n2=100, is equivalent to long wires being limited to 100um in length. In addition, two-dimensional fracturing is performed. 2D fracturing is good for metal plates.

### Examples

#### Example 1

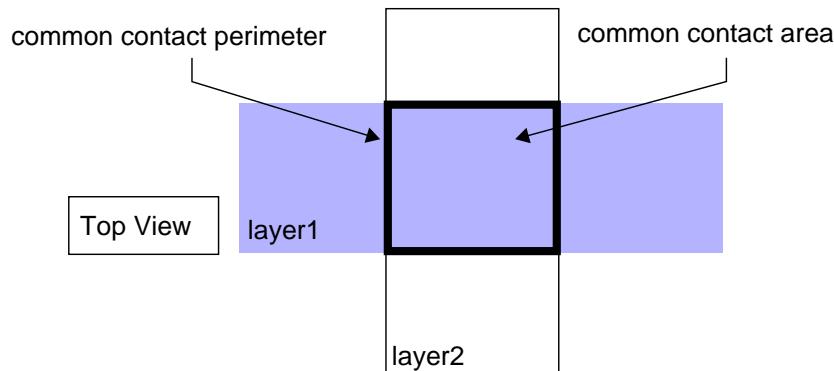
```
CONNECT metal poly BY contact  
RESISTANCE CONNECTION metal poly [0.05 0.1]  
// The next line will give a PEX5 error because of layer order:  
//RESISTANCE CONNECTION poly metal [0.05 0.1]
```

#### Example 2

When two layers connect directly, as with the following lines, the common area is large as shown in [Figure 4-316](#).

```
CONNECT layer1 layer2  
RESISTANCE CONNECTION layer1 layer2 [0.83 0]
```

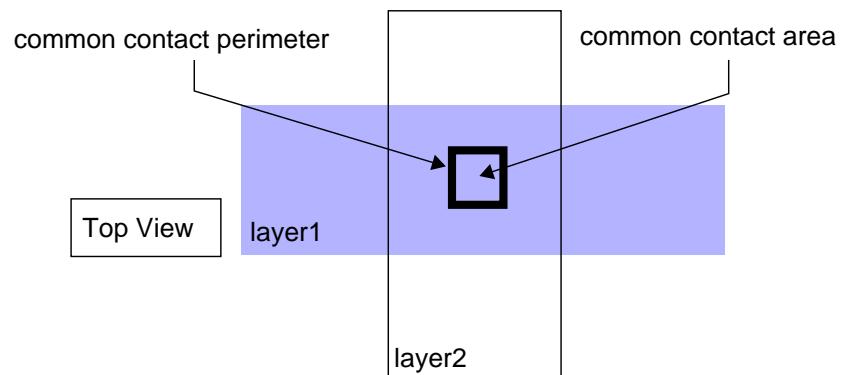
**Figure 4-316. Example Two-Layer Resistance Connection**



#### Example 3

When two layers are connected through a third layer, usually a via, the common area is only that which is common to all involved layers as shown in [Figure 4-317](#).

```
CONNECT layer1 layer2 BY via  
RESISTANCE CONNECTION layer1 layer2 [0.83 0]
```

**Figure 4-317. Example Connect By Resistance Connection**

# Resistance Device\_Seed

Parasitic extraction

**RESISTANCE DEVICE\_SEED *pin\_layer* *seed\_layer* '[' *device\_layer\_resistance* **0** ']**

## Parameters

- ***pin\_layer***  
A required pin layer defined for the device.
- ***seed\_layer***  
A required seed layer defined for the device.
- **[*device\_layer\_resistance* **0**]**  
A pair of required strings that must be enclosed in brackets ( [ ] ). The strings are described as follows:
  - device\_layer\_resistance*** — A non-negative real number, which is in units of resistance squared, which specifies the inverse area proportionality constant.
  - 0** — A required number that must always be 0.

For more information about units of resistance, refer to the [Unit Resistance](#) statement.

## Description

Defines the proportionality constant for computing the resistance through a gate. Use this statement in place of the [Resistance Sheet](#) statement to specify resistance on device layers. *Seed\_layer* must derive its connectivity from *pin\_layer*.

## Examples

### Example 1

The Resistance Device\_Seed statement is used in addition to the standard statements describing resistance and connection.

```
CONNECT metall1 poly BY contact
RESISTANCE SHEET metall1 [ 0.03 0 ]
RESISTANCE DEVICE_SEED poly pgate [ 1 0 ]
```

### Example 2

If the device region has been cut out to prevent double counting of parasitic capacitance (sometimes referred to as cookie-cutting), you need to explicitly connect the device region in order to calculate resistance. In those cases, the device layers must appear in a [Connect](#) statement as shown in this example:

```
CONNECT metall1 poly BY contact
CONNECT poly pgate
RESISTANCE SHEET metall1 [ 0.03 0 ]
RESISTANCE DEVICE_SEED poly pgate [ 2 0 ]
```

# Resistance Rho

Parasitic extraction

**RESISTANCE RHO layer ‘[*n1 n2 [TC1 TC2]*’]**

## Parameters

- *layer*

A required original layer.

- [*n1 n2*]

A pair of required strings that must be enclosed in brackets ( [ ] ). The strings are described as follows:

***n1*** — A non-negative real number, corresponding to the bulk resistivity of the layer for its nominal thickness

***n2*** — A non-negative real number, which is in units of length, that specifies the maximum length of any element. The default is 100 microns.

- *TC1 TC2*

A pair of optional strings that are required only if you are modifying the resistance extracted during the pdb step with the [PEX Thickness EQN](#) statement.

### Note



As of the 2006.4 release, TC1 and TC2 are deprecated. Use [PEX Resistance Parameters](#) instead. All PEX Resistance Parameters settings override any [Resistance Sheet](#) settings.

These parameters are floating point numbers specifying the first- and second-order temperature coefficients of resistance, in degrees Celsius. You can define the temperature coefficients on a layer by layer basis, with each layer and netlist having a different set of temperature coefficients.

## Description

This statement specifies the rho (bulk resistivity) of a layer, and is used for computing resistance variation due to in-die width and thickness variation. This statement can be used alone, or in conjunction with the Parasitic Variation ... RHO statement.

Resistance Rho overrides Resistance Sheet. The ***n2*** parameter is set by the following order of precedence from highest to lowest:

1. [Parasitic Variation](#)
2. [PEX Resistance Parameters](#)
3. Resistance Rho
4. [Resistance Sheet](#)

### **Caution**



The Resistance Rho statement requires a way to calculate thickness. Therefore, if you use Resistance RHO, you must also specify either the Parasitic Variation Thickness or PEX Thickness Nominal statement for the layer.

---

For a description of the Calibre xRC in-die variation process, including information on when and when not to use it, refer to the [\*Calibre xRC User's Manual\*](#).

### Examples

```
RESISTANCE RHO metall1 [0.033333 0]
```

# Resistance Sheet

Parasitic extraction

## RESISTANCE SHEET *layer1* '['*n1 n2* [*TC1 TC2*]']'

### Parameters

- *layer1*

A required original layer or a derived polygon layer.

- [*n1 n2*]

A pair of required strings that must be enclosed in brackets ( [ ] ).

***n1*** — A non-negative real number in units of resistance per square, which specifies the proportionality constant.

***n2*** — A non-negative real number in units of length, which specifies the maximum length of any element. The default is 0 microns.

For more information about units of resistance, refer to the [Unit Resistance](#) statement.

- *TC1 TC2*

A pair of optional strings that are required only if you are modifying the resistance extracted during the pdb step with the [PEX Thickness EQN](#) statement.

**Note**



As of the 2006.4 release, TC1 and TC2 are deprecated. Use [PEX Resistance Parameters](#) instead. All PEX Resistance Parameter settings override any Resistance Sheet settings.

These parameters are floating point numbers specifying the first- and second-order temperature coefficients of resistance, in degrees Celsius. You can define the temperature coefficients on a layer by layer basis, with each layer and netlist having a different set of temperature coefficients.

### Description

Defines the proportionality constants for computing a resistance for current flowing within a conduction layer.

The input layer must appear in a [Connect](#) BY operation. You cannot use *layer1* as a contact layer in any Connect operation.

The ***n2*** setting specifies the maximum length of a segment. After fracturing, rectangular conductive shapes aligned along the design's axes on *layer1* with an edge longer than ***n2*** are split into an odd number of smaller rectangles of equal size. These are then represented by a parasitic resistor. Angled and non-rectangular shapes are not split, regardless of size. Generally, 0 microns ensures that interconnect wires are only fractured along their length and not along their width. Setting ***n2*** to a value less than the width of standard interconnect causes excessive fracturing which leads to longer run times.

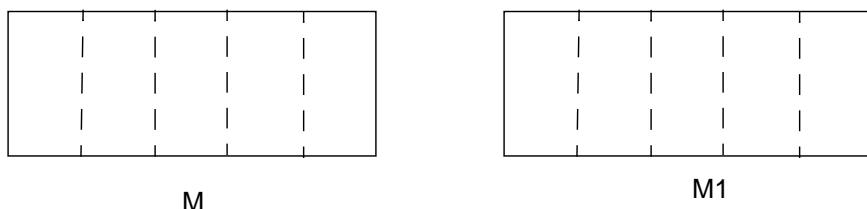
**Examples**

For an extraction with a rulefile containing these rules:

```
CONNECT M1 poly BY contact
RESISTANCE SHEET M2 [ 0.03 0 0.0028 0.0035 ]
RESISTANCE SHEET M1 [ 0.03 80 ]
```

a rectangle that is 320 by 80 microns would be split into resistors as shown in [Figure 4-318](#). Notice that the segments are the same for M2 and M1. Although n2 for the M2 rule is 0 which resets to 100 microns, the largest whole number less than 100 that divides 320 into an odd number of segments is 64. Therefore, the shape is divided into 5 equal parts with a length of 64 on both layers.

**Figure 4-318. Resistance Sheet Fracturing**



If the rectangle had been 300 by 80 microns, M2 would have been divided into 3 regions of 100 by 80 microns each. M1 would still be at 5 regions, each 60 by 80 microns.

# Resolution

Specification statement

## RESOLUTION {*s* | *x* *y*}

### Parameters

- *s* | *x* *y*

A required positive integer, or pair of positive integers, that specifies the layout grid (also called the user grid) step size in *database units*. Specifying one value, *s*, causes the grid step to be square (equal in x- and y-directions). Specifying *x* *y* indicates that *x* is the x-direction grid step and *y* is the y-direction grid step. The default behavior is *s* equal to one database unit, if you do not include this statement in the rule file.

### Description

Defines the layout grid step size. This is the multiple of database units on which any original geometry is to be aligned. This statement is optional and, if specified, can appear once in a rule file. It is independent of any other settings.

The primary use of the Resolution statement is to allow off-grid original polygons to be flagged. Note that this statement has no effect upon the [Snap](#) or [Offgrid](#) layer operations.

See also [Drawn Offgrid](#), [Flag Offgrid](#), [Snap Offgrid](#), [Layer Resolution](#), and [Precision](#).

### Examples

The following grid step size requires original shapes to align at 250, 500, 750, or 1000 database units. This is assuming the layout's physical precision matches the rule file Precision statement.

```
PRECISION 1000      // 1000 database units per user-unit.  
RESOLUTION 250  
// Alignment points of original polygons must be every 0.25  
// user units.
```

## RET NMDPC

Layer operation

**RET NMDPC** [*action\_layer*] *fabric\_layer*  
[*action\_layerN...*] **FILE** *parameter\_block\_name* **MAP** *name*

### Parameters

- ***action\_layer***

An optional argument specifying an action layer. This layer can be a polygon type layer, an edge type layer, or an error type layer. Additional action layers can be specified after the ***fabric\_layer*** argument. For more information on these layer types, refer to the “[Layers](#)” section of the *Calibre Verification User’s Manual*.

- ***fabric\_layer***

A required argument specifying the input layer. This is the layer to be decomposed.

- ***action\_layerN***

An optional list of supplementary action layers. Each of the layers specified in this list must be a polygon type layer, an edge type layer, or an error type layer. For more information on these layer types, refer to the “[Layers](#)” section in the *Calibre Verification User’s Manual*.

- **FILE** *parameter\_block\_name*

A required keyword and argument that specifies the name of a parameter block defined within the SVRF rule file, using the LITHO FILE command block. A separate setup file can also be specified.

For more information on the syntax of the LITHO FILE (for nmDP-C) command, refer to the “[LITHO FILE \(for nmDP-C\)](#)” section of the *Calibre Double Patterning User’s Manual*.

- **MAP** *name*

A required keyword followed by the associated argument indicating the layer containing the output to be returned. The layer name can be conflict, mask0, or mask1.

When multiple RET NMDPC operations are present and differ only with respect to their MAP name, the Calibre nmDRC hierarchical engine performs the RET operation once, generating multiple output layers simultaneously.

### Description

The RET NMDPC command is a tool argument (RET NMDPC) to the Calibre RET batch operation. It is used to invoke Calibre nmDP-C and produce the layer output, either a pitch conflict layer, action conflict layer, or mask layers.

### Examples

#### Example 1

A pitch conflict occurs when the contacts cannot be separated into two masks based on the specified pitches. In the following example, if no pitch conflicts are detected the layout will be

split and mask0 and mask1 will be output. However, if conflicts are found, then pitch\_layer\_conflict will capture these conflicts at the lowest level of hierarchy, and this layer will be output.

```
pitch_layer_conflict = RET NMDPC input_layer FILE mypitches MAP conflict
mask0 = RET NMDPC input_layer FILE mypitches MAP mask0
mask1 = RET NMDPC input_layer FILE mypitches MAP mask1
```

### Example 2

An action layer conflict occurs when the contacts cannot be separated into two masks based on the specified layers. In this example, if no action layer conflicts are detected the layout will be split and mask0 and mask1 will be output. However, if conflicts are found, then action\_layer\_conflict will capture these conflicts at the lowest level of hierarchy, and this layer will be output.

```
action_layer_conflict = RET NMDPC action_layer input_layer
    optional_action_layer FILE action_file MAP conflict
mask0 = RET NMDPC action_layer input_layer
    optional_action_layer FILE action_file MAP mask0
mask1 = RET NMDPC action_layer input_layer
    optional_action_layer FILE action_file MAP mask1
```

## RET PXOPC

Layer operation

**RET PXOPC** *layer [layer ...] FILE {filename | name | '['  
    *inline\_file*  
    ']}' } MAP name*

Used only in Calibre pxOPC applications.

### Parameters

- ***layer***

A required original layer, derived polygon layer, or edge layer upon which to perform Calibre pxOPC operations. You can specify one or more *layers* in one statement. The layer statements in the litho setup file determine the function of the *layers*.

- ***FILE {filename | name | '['  
    *inline\_file*  
    ']}' }***

Required keyword, followed by one of the following:

A *filename* indicating the path to the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).”

The *name* parameter of a [Litho File](#) specification statement. The setup file is specified in the Litho File statement.

An *inline\_file* between brackets ([ ]). Characters within the brackets on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- ***MAP name***

A required keyword, followed by the name of a layer from the setup file. This layer is then mapped to the layer in the rule file for output.

### Description

Specifies to run Calibre pxOPC during Calibre nmDRC or nmDRC-H. Calibre pxOPC performs full mask optimization, including OPC and SRAF insertion and optimization, upon the *layers* you specify. The command takes in a target layer and outputs a new layer with OPC-corrected shapes and optimized SRAFs.

This command can be specified any number of times in your rule file.

Refer to the [Calibre pxOPC User’s and Reference Manual](#) for Calibre pxOPC process and procedures. The manual also describes the setup file commands.

## Examples

### Example 1 - Layer Passing

The following example shows how layers are mapped between the RET PXOPC command and the setup file. (The contents of the setup file show only the parts relevant to this example. A working setup file would also specify several other settings.)

```
L1 = RET PXOPC m1a m1_smth FILE "setup.in" MAP L1
L1 {COPY L1} DRC CHECK MAP L1 20
...
LITHO FILE setup.in [
  ...
    layer orig hidden clear // "global" part of setup file; pxopc layers
    layer smoothed hidden clear //hidden to just pass through
  ...
  pxopc_options px.options { //pxopc-specific commands
    layer orig CORRECTION clear
    layer smoothed TARGET clear
  ...
}
...
setlayer mask_out = pxopc orig smoothed MAP orig OPTIONS px.options
...
]
```

The RET PXOPC command passes the m1a and m1\_smth layers to the setup file, where they are mapped to the orig and smoothed layers respectively. These names are then used throughout the setup file.

### Example 2 - Alternate Forms

In Example 1, the RET PXOPC command assigns the output (mask\_out) to a layer available to the rest of Calibre (L1). This makes it available for more processing. Alternatively, the RET PXOPC statement might be enclosed in a rule check and have its output written directly to a GDS or results database, as shown here:

```
L1 {RET PXOPC m1a m1_smth FILE "setup.in" MAP L1} DRC CHECK MAP L1 20
...
```

The line shown here would replace the two lines from before:

```
L1 = RET PXOPC m1a m1_smth FILE "setup.in" MAP L1
L1 {COPY L1} DRC CHECK MAP L1 20
```

### Example 3 - Creating Target Layer In Setup File

Although the target layer is commonly passed in as a second layer in the RET PXOPC statement, it does not have to be. The target layer can be created inside the setup file as shown in the following code:

```
L1 {RET PXOPC m1a FILE "setup.in" MAP L1} DRC CHECK MAP L1 20
...
LITHO FILE setup.in [
...
  layer orig hidden clear
...
  pxopc_options px.options { //pxopc-specific commands
```

## RET PXOPC

---

```
layer orig CORRECTION clear
layer smoothed TARGET clear
...
}
setlayer smoothed = curve orig      //This creates a smoothed target
...
setlayer mask_out = pxopc orig smoothed MAP orig OPTIONS px.options
...
]
```

# RET SBAR

Layer operation

**RET SBAR** *layer [layer ...] FILE {filename | name | '['  
    *inline\_file*  
    ']}' } MAP *name**

Used only in Calibre nmSRAF applications.

## Parameters

- ***layer***

One or more required input layers upon which to perform Calibre nmSRAF operations. All layers are processed in the order listed.

- **FILE {filename | name | '['  
    *inline\_file*  
    ']}' }**

Required keyword, followed by one of the following:

A *filename* indicating the path to the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#).“

The *name* parameter of a [Litho File](#) specification statement. The setup file is specified in the Litho File statement.

An *inline\_file* between brackets ([ ]). Characters within the brackets on the same line as them are ignored. Inline files contain setup instructions for OPC and span multiple lines. The instructions may not use environment variables.

- **MAP *name***

A required keyword, followed by the name of a layer from the setup file. This layer is then mapped to the layer in the rule file for output.

## Description

The RET SBAR command takes a target layer and uses the geometries on this layer and the templates specified in the litho file to output a new layer of SRAFs. This command can be specified any number of times in your rule file. Refer to the [Calibre nmSRAF User’s and Reference Manual](#) for more information on Calibre nmSRAF processes and procedures.

## Examples

### Example 1

The following example shows how layers are mapped between the RET SBAR command and an inlined setup file. (The contents of the setup file show only the parts relevant to this example. A working setup file would also specify several other settings.)

```
1_pre      = RET SBAR V1_target FILE cn_setup_1 MAP 1_pre
1_field    = RET SBAR V1_target FILE cn_setup_1 MAP 1_field
1_hard     = RET SBAR V1_target FILE cn_setup_1 MAP 1_hard
1_conflict = RET SBAR V1_target FILE cn_setup_1 MAP 1_conflict
1_processed = RET SBAR V1_target FILE cn_setup_1 MAP 1_processed
1_iso      = RET SBAR V1_target FILE cn_setup_1 MAP 1_iso

LITHO FILE cn_setup_1 [
    layer V1_target

    setlayer 1_pre      = cnsraf V1_target ...
    setlayer 1_field    = cnsraf V1_target ...
    setlayer 1_hard     = cnsraf V1_target ...
    setlayer 1_conflict = cnsraf V1_target ...
    setlayer 1_processed = cnsraf V1_target ...
    setlayer 1_iso      = cnsraf V1_target ...
]

```

The RET SBAR commands are identical except for the output layer specified by the MAP keyword. The commands pass the V1\_target layer to the setup file, where it is mapped to the mask layer.

In the setup file, the commands “cnsraf...” generate a layer. The first one is written to a layer named “1\_pre” and the second to “1\_field” and so on. The MAP keyword in the RET SBAR statement identifies the intended matching SVRF statement.

In this example, the RET SBAR statements then assign the output to a layer available to the rest of Calibre. This makes it available for more processing. Alternatively, the RET SBAR statement might be enclosed in a rule check and have their output written directly to a GDS or results database.

### Example 2

This example uses an sblayer called “my\_sblayer” and an offset\_layer called “my\_offset\_layer”. The RET SBAR statement then assigns the output to layer called “2\_iso” and makes this layer available to the rest of Calibre.

```
2_iso = RET SBAR V2_target my_sblayer my_offset_layer FILE cn_setup_2
        MAP 2_iso

LITHO FILE cn_setup_2 [
    layer V2_target
    layer my_sblayer
    layer my_offset_layer

    setlayer 2_iso = cnsraf V2_target
                    sblayer my_sblayer offset_layer my_offset_layer ...
]

```

# RET Sraf\_Fill

Layer operation

**RET SRAF\_FILL** *target\_layer sraf\_layer [offsetlayer] FILE file*

## Parameters

- ***target\_layer***  
A required argument specifying the target layer.
- ***sraf\_layer***  
A required argument specifying the SRAF layer.
- ***offsetlayer***  
An optional argument specifying the name of a layer that defines areas where scattering bars cannot be created. No SRAFs will be placed within the minoffset distance (specified in the cnsraf command) from the specified *offsetlayer*.
- ****FILE** *file***  
A required argument specifying a pathname to a file, a litho file definition in the SVRF rule file, or parameter definitions inside a pair of square brackets.

The following parameter definitions can be specified in *file*:

- **maxsboffset** *value*  
An optional keyword and parameter specifying the maximum distance from the target layer polygons where additional SRAFs may be generated. This is a floating point number and has no default value.
- **minsboffset** *value*  
An optional keyword and parameter specifying the minimum distance from the target layer polygons where SRAFs are not allowed. This is a floating point number and has no default value.
- **minsbspace** *value*  
An optional keyword and parameter specifying the distance from the SRAF layer polygons where SRAFs are not allowed. The newly generated SRAFs will maintain a distance of minsbspace from each other and from the existing SRAF layer polygons. This is a floating point number and has no default value.
- **sbwidth** *value*  
An optional keyword and parameter specifying the width of newly generated SRAFs. This is a floating point number and has no default value.

- o maxsblength *value*

An optional keyword and parameter specifying the maximum length of the newly generated SRAFs. The minimum length of SRAFs generated by the RET SRAF\_FILL command is sbwidth (square SRAFs). This is a floating point number and has no default value.

- o delta *value*

An optional keyword and parameter specifying the step size by which to increment the SRAF length when trying to generate SRAFs of maximum length. Note that there is a 40 step limit between sbwidth and maxsblength. This is a floating point number and has no default value.

## Description

This command is used to eliminate holes in SRAF layers generated by the [OPCSBAR](#) or [cnsraf](#) commands.

## Examples

### Example 1

```
seed = RET SRAF_FILL target sbar FILE "./mysetup.file"
```

### Example 2

```
LITHO FILE myfile.ret [
    maxsboffset 0.23
    minsboffset 0.07
    minsbspace 0.05
    sbwidth     0.03
    maxsblength 0.50
]
seed = RET SRAF_FILL target sbar FILE myfile.ret
```

### Example 3

```
seed = RET SRAF_FILL target sbar FILE [
    maxsboffset 0.22
    minsboffset 0.07
    minsbspace 0.05
    sbwidth     0.025
    maxsblength 0.50
]
```

# Rotate

Layer operation

**ROTATE** *layer* **BY** *angle*

## Parameters

- ***layer***  
A required original or derived polygon layer.
- **BY *angle***  
A required keyword set, where *angle* is a floating-point number that specifies the degrees of rotation. Positive values yield a counter-clockwise rotation.

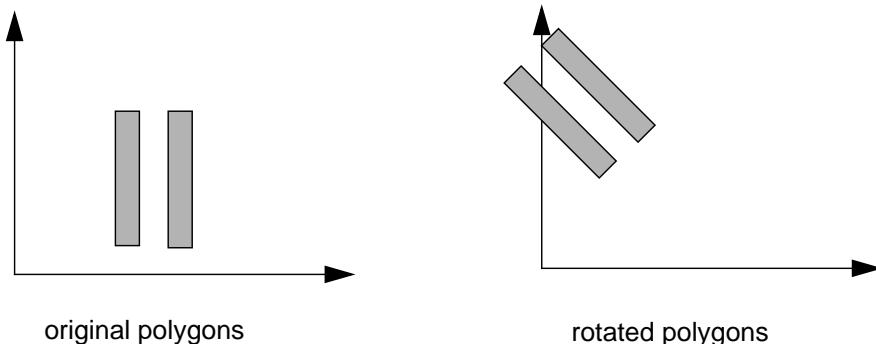
## Description

Creates a derived polygon layer by rotating all polygons on the input layer about the database origin by the given angle.

This is a non-node-preserving layer constructor and is performed flat in hierarchical applications; it can be very slow.

## Examples

```
rule { ROTATE poly BY 45 }
```



## Sconnect

Connectivity extraction

Syntax 1:

**SCONNECT** *upper\_layer* *lower\_layer* [*lower\_layer* ...] {**BY** *contact\_layer*} [**LINK** *name*] [**DIRECT**] [**MASK**]

Syntax 2:

**SCONNECT** *upper\_layer* *lower\_layer* [**LINK** *name*] [**ABUT ALSO**] [**DIRECT**] [**MASK**]

### Summary

Establishes one-way connections from an upper layer to lower layers through a contact layer (Syntax 1). Establishes one-way connections from an upper layer to a lower layer (Syntax 2). Primarily used for detection of soft connections, and in conjunction with [LVS Softchk](#) or the lvs::softchk compile-time TVF function.

### Parameters

- ***upper\_layer***

A required original layer or layer set, or a derived polygon layer. If this layer is a derived layer, it must be defined in the global scope of the rule file.

- ***lower\_layer***

A required original layer or layer set, or a derived polygon layer. You can specify a maximum of 32 of these layers in a single Syntax 1 statement. You can specify only one *lower\_layer* in a Syntax 2 statement. Any layer specified as a *lower\_layer* cannot simultaneously be a *contact\_layer* in any Sconnect operation. Any *lower\_layer* may be specified as a *lower\_layer* in a different Sconnect operation. If this layer is a derived layer, it must be defined in the global scope of the rule file.

- **BY *contact\_layer***

A required keyword set used with Syntax 1, where *contact\_layer* is an original layer or layer set, or a derived polygon layer. The keyword **BY** must always precede the name of this contact layer. If you specify more than one *contact\_layer*, the first layer is used and the rest are ignored. If this layer is a derived layer, it must be defined in the global scope of the rule file.

- **LINK *name***

An optional keyword set, where *name* indicates a node name. Floating polygons receive the node ID of the electrical node having the specified *name* in the top-level cell. The *name* must already exist in the layout database in order for it to be applied. Floating polygons are polygons on any specified *lower\_layer* that are not connected to any *upper\_layer* polygons.

- **ABUT ALSO**

A optional keyword that specifies that abutment is considered to constitute overlap in this operation. Applies to Syntax 2 only.

- **DIRECT**

An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.

- **MASK**

An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

## Description

Passes connectivity from the ***upper\_layer*** polygons to ***lower\_layer*** polygons. Connections are unidirectional; nodal information is passed from ***upper\_layer*** to ***lower\_layer***, but not in the other direction.

In Syntax 1, connectivity information is passed from ***upper\_layer*** objects to ***lower\_layer*** objects, through ***contact\_layer*** objects. The ***lower\_layer*** objects must have overlapping area common to both ***upper\_layer*** and ***contact\_layer*** objects. That is, polygons from all three of these layers must have a mutual intersection in order for connectivity to be passed. Polygon abutment does not constitute a valid connection for this syntax. The ***upper\_layer*** and ***contact\_layer*** objects are connected, and the first available ***lower\_layer1*** through ***lower\_layerN*** object in that order, is also connected.

If more than one ***lower\_layer*** is specified using Syntax 1, then the first object found from among ***lower\_layer1*** through ***lower\_layerN*** (in that order) at a given location participates in the connection; no other layers from ***lower\_layer1*** through ***lower\_layerN*** become connected. This shielding behavior is similar to what occurs in a [Connect BY](#) statement, except that Sconnect BY typically pushes connections down to the hierarchical level that the connections originate from. However, Sconnect BY can cause the same side-effects from shielding as occur in Connect BY. In general, shielded Sconnect statements should be avoided. For further discussion of this topic, refer to the “[Shielding](#)” section of the *Calibre Verification User’s Manual*.

The example in [Figure 4-12](#) shows similar behavior to the Syntax 1 Sconnect operation. Recall that Sconnect passes connectivity information in one direction only. This is different from the [Connect](#) statement behavior.

In Syntax 2, connectivity is passed from ***upper\_layer*** to ***lower\_layer*** objects. Polygon abutment constitutes a valid connection in Syntax 2 if you use ABUT ALSO. Polygons touching at corners do not constitute abutment.

If two or more electrical nodes on ***upper\_layer*** form soft connections to the same ***lower\_layer*** polygon, then the electrical node that contains the largest number of polygon vertices (including contacts) is chosen. Note that shapes are merged per layer and, in hierarchical operation, shapes at different levels of hierarchy are not merged. The choice of largest node is made across all Sconnect BY and Sconnect operations that form connections to the particular ***lower\_layer***

polygon. This conflict resolution method is modified slightly when Sconnect chains are present; refer to “[Chained Sconnect Operations](#)” for further information.

Polygons on *lower\_layer* that are not connected to any *upper\_layer* in any Sconnect operation are called *floating*. Floating polygons receive unique node numbers if one of these is true:

- The optional keyword LINK is not used.
- The optional keyword LINK is used, but no node exists having the specified *name*. A warning is issued in this case.

Conflicting connections from Sconnect operations can be reported with [LVS Softchk](#). See “[LVS Softchk Debug Method](#)” in the *Calibre Verification User’s Manual* for more details on how to debug LVS Softchk errors.

In some cases, it may make better sense to use the count of contacts shared between *upper\_layer* and *lower\_layer* polygons on a net rather than the number of vertices on a net to resolve a stamping conflict. This method involves using the lvs::softchk function. See “[Reporting Soft Connections Using Contact Counts](#)” in the *Calibre Verification User’s Manual* for more information.

[LVS Report Option S](#) turns on detailed reporting of Sconnect conflicts in the session transcript. This option is ignored in the nmDRC applications.

In general, Sconnect is preferred to using [Stamp](#).

The connectivity model is built from all Connect and Sconnect statements in the rule file *after* all the derived layers required for the Connect and Sconnect statements have been generated. The connectivity model is constructed in a single step *before any operation that requires or passes connectivity*. (The only exception is when [DRC Incremental Connect YES](#) is specified.) For example:

```
CONNECT layer1
SCONNECT layer1 layer2

layer2 = layer3 AND layer4 // this layer is generated before connectivity
```

Here, the AND operation is performed before connectivity is established on layer2. If a node-preserving operation derives a layer that appears in a Connect or Sconnect statement, that operation does not pass connectivity.

This behavior can have unexpected side-effects if connectivity is not managed carefully. For example:

```
CONNECT layer1
CONNECT layer2
SCONNECT layer1 layer3

layer3 = layer2 AND layer4
```

Again, the AND operation is evaluated before connectivity is established. Node IDs are present on layer3 (assuming layer1 interacts with layer3) because of the SCONNECT statement, but are not passed by the AND operation because there is no connectivity when this operation is performed (otherwise layer2 and layer3 would be shorted together). This can cause unexpected

side-effects such as unattached label warnings involving connectivity layers and net names not being established in certain cases.

A good practice to follow is to use Connect and Sconnect statements for a relatively small set of layers. Then you can use node-preserving operations to pass connectivity to derived layers as needed. Careful management of derived layers appearing in connectivity statements is advised. Node-preserving operations are discussed in detail in the [Calibre Verification User's Manual](#).

## Requirements and Restrictions

- The ***upper\_layer*** must appear in at least one [Connect](#) operation, or as a ***lower\_layer*** in at least one Sconnect operation.
- A ***lower\_layer*** cannot appear in Connect operations or be derived using connectivity-dependent operations.
- A ***lower\_layer*** in one Sconnect operation cannot serve as a ***contact\_layer*** in the same or another Sconnect operation.

For example, the following examples are not allowed:

```
SCONNECT D C
SCONNECT A B BY C // not allowed
SCONNECT A C BY C //not allowed
```

- You cannot use a Mask mode Sconnect operation in the rule file when [DRC Incremental Connect YES](#) is specified. This situation causes a compilation error.
- The ***contact\_layer*** cannot be specified in two Sconnect BY operations of different priorities. Refer to the section “[Chained Sconnect Operations](#)” for information on how the tool determines Sconnect priorities.
- Using layers with large polygons as contact layers can lead to unexpected results. Separate polygons on ***upper\_layer*** are considered part of the same net if they share a common ***contact\_layer*** polygon. Deriving a different ***contact\_layer***, or using Syntax 2, is preferable in such cases.

For example, assume you are trying to detect soft connections through your nwell layer. Consider this code:

```
CONNECT MET1 NTAP BY CONT // NWELL TIE-DOWNS
X = COPY NWELL
SCONNECT NTAP NWELL BY X
LVS SOFTCHK NWELL CONTACT
```

This example will not find soft connections from NTAP through NWELL. The reason is any NTAP polygons with supposedly different connectivity, which lie within a single polygon on X, are all considered to be on the same net by Sconnect. A better way is not to use a large contact layer:

```
CONNECT MET1 NTAP BY CONT
SCONNECT NTAP NWELL
LVS SOFTCHK NWELL CONTACT
```

## **C chained Sconnect Operations**

A chained Sconnect operation occurs when the *upper\_layer* of an Sconnect operation receives a node number created through a Connect or Sconnect operation. For example:

```
CONNECT A AA  
SCONNECT A B  
SCONNECT B C // This use of layer B is allowed
```

The tool follows two criteria when determining if chained Sconnect operations are valid; the first of which covers the majority of applications.

- If each *lower\_layer* in a Sconnect operation receives node numbers from only one Sconnect operation, the Sconnect operations are allowed.
- If a layer received node numbers from more than one Sconnect operation, the tool uses the following procedure:
  - Each layer is assigned a number, referred to as *priority*.
  - The *priority* of all Connect layers is 0.
  - If the priority of an *upper\_layer* of a Sconnect operation is N, then all of the *lower\_layers* in the same operation have a priority of N+1.

The chained Sconnect operations are valid if the priority assignment is unambiguous. In other words, when a layer appears as a *lower\_layer* in several Sconnect operations, it should receive the same priority in all of them, which implies that the *upper\_layers* of these Sconnect operations all have the same priority.

- Contact layers cannot be shared between Sconnect operations of different priorities. (The priority of an Sconnect operation is defined as the priority of its *lower\_layer* parameters). For example, the following is not valid because the contact layer CONT is shared between two Sconnect operations of different priority in a chain:

```
CONNECT A AA          // A has priority 0.  
SCONNECT A B BY CONT // B has priority 1.  
SCONNECT B C BY CONT // C has priority 2. Error.
```

The correct way to write this Sconnect chain is this:

```
SCONNECT A B BY CONT  
CONT1 = COPY CONT  
SCONNECT B C BY CONT1
```

In addition, contact layers of Connect operations cannot be shared with chained Sconnect operations of priority greater than 1. Only highest-priority Sconnect operations can share contact layers with Connect operations. For example, this is valid:

```
CONNECT A B BY CONT  
SCONNECT B C BY CONT // Priority 1.
```

This is not valid:

```
CONNECT A B BY CONT
SCONNECT B C // Priority 1.
SCONNECT C D BY CONT /* Priority 2. Cannot share contact layer with
CONNECT.*/
```

Looped Sconnect operations are invalid, as shown in the following rule file example:

```
CONNECT A AA
SCONNECT A B //priority of B is 1
SCONNECT B C
SCONNECT C D
SCONNECT D B //priority of B is 4 (ambiguous)
```

The next rule file example shows an invalid Sconnect chain, which is not a loop:

```
CONNECT A AA
SCONNECT A B
SCONNECT B C //priority of C is 2
SCONNECT A C //priority of C is 1 (ambiguous)
```

This is permissible:

```
CONNECT A AA
SCONNECT A B //priority of B is 1.
SCONNECT B C //priority of C is 2.
A1 = COPY A
SCONNECT A A1 //priority of A1 is 1.
SCONNECT A1 C //This is allowed.
```

The tool issues an error EXT13 when the rule file compiler detects an invalid Sconnect chain.

## Conflict Resolution for Sconnect Chains

The tool modifies the resolution of Sconnect conflicts when Sconnect chains are present; this is the procedure:

- Each Sconnect operation is assigned a priority value, which is the priority of its *upper\_layer* in the previously-described procedure.
- The tool executes Sconnect operations in order of their priorities, starting from 0. After the tool executes all Sconnect operations with a given priority and resolves any conflicts, node sizes (as measured by polygon vertex counts) are computed again. These new node sizes are then used for conflict resolution in the next priority group of Sconnect operations.

The following rule file example illustrates how priorities are assessed:

```
CONNECT A AA
SCONNECT A B //priority 0.
SCONNECT B C //priority 1.
SCONNECT B D //priority 1.
```

- The tool executes operation SCONNECT A B first.

## Sconnect

---

If a conflict between two nodes exists at this stage, then the conflict is resolved based on node sizes as determined by the Connect operation.

- The tool recomputes node sizes to include polygons stamped by this Sconnect operation.
- The tool executes operations SCONNECT B C and SCONNECT B D. Conflict resolution now uses the new updated node sizes.

The rule file compiler verifies that Sconnect chains are properly specified and any errors are reported.

As an automatic step, trivial pins are removed following connectivity extraction. The following definitions are provided before describing how trivial pins are handled:

A non-trivial net has:

- one or more polygons OR
- text or port text

A non-trivial pin is defined as:

- a pin of an LVS Box cell OR
- a pin of the top cell OR
- on the minimal path between two or more non-trivial nets or pins.

A trivial pin is any pin that does not conform to the previous definition.

After all non-trivial nets and pins are identified, the remaining trivial pins are removed.

## Examples

### Example 1

Common soft connection check:

```
CONNECT M1 welltie  
// perform soft connection check of welltie to well.  
SCONNECT welltie well  
LVS SOFTCHK well UPPER
```

### Example 2

Numerous examples of soft-connection checks for both nmDRC and nmLVS are discussed under “Soft Connection Checks” in the [Calibre Verification User’s Manual](#).

# Shift

Layer operation

**SHIFT** *layer* **BY** *x* *y*

## Parameters

- ***layer***  
A required original layer or a derived polygon layer.
- **BY *x* *y***  
A required keyword set, where the strings *x* and *y* are real numbers in user units, which specify how much to shift the *layer* polygons. The *x* and *y* values specify the distance along the x-axis and y-axis, respectively, to move the polygons. Negative numbers must be enclosed in parentheses.

## Description

Shifts all polygons on the specified layer by the specified *x* and *y* values. That is, every polygon vertex  $(x_0, y_0)$  on *layer* is mapped to  $(x_0 + x, y_0 + y)$ .

The hierarchical form of the Shift operation is done by promoting geometry from cells where at least two distinct placements of the cell, in the flat view of the design, have differing rotation- or reflection-transformation components. Promotion stops in a cell that, in the flat view, has a consistent rotation- or reflection-transformation component in all of its placements. The Shift operation can then be performed in that cell and the result will be correct from the flat view.

Note that hierarchical Shift operation can be slow in some designs, for example, where a cell is extremely large. Performing the hierarchical Shift operation on this large cell flattens it to the top-level cell. However, in many cases, the hierarchical Shift operation for certain design layers can provide better performance than flat implementation, if the degree of hierarchical degradation of the result layer is minimal.

This operation does not check coordinate space overflow. This means if your runtime database extent (see [Extent](#) for a complete discussion) together with the Shift values you specify exceed the limits of coordinate space size on your host machine, there is no warning.

## Examples

### Example 1

The following statement in a rule check shifts via polygons one unit to the right and one unit up:

```
SHIFT via BY 1 1
```

### Example 2

The following statement shifts metal polygons two units to the right and three units down:

```
SHIFT metal BY 2 (-3)
```

**Example 3**

Note that the following statement causes a compilation error, because it is interpreted as Shift metal By -1, which is incomplete syntax:

```
SHIFT metal BY 2 -3
```

# Shrink

Layer operation

**SHRINK** *layer* [RIGHT BY *value*] [TOP BY *value*] [LEFT BY *value*] [BOTTOM BY *value*]

## Parameters

- *layer*

A required original or derived polygon layer, or derived edge layer.

At least one of the following must be specified:

- RIGHT BY *value*

Keyword set that instructs the tool to contract, toward the interior, the *right* edge(s) of objects on the input *layer* by the specified *value*, which is a non-negative floating-point number in user units. Refer to the description for how the tool determines which edge is considered a *right* edge

- TOP BY *value*

Keyword set that instructs the tool to contract, toward the interior, the *top* edge(s) of objects on the input *layer* by the specified *value*, which is a non-negative floating-point number in user units. Refer to the description for how the tool determines which edge is considered a *top* edge

- LEFT BY *value*

Keyword set that instructs the tool to contract, toward the interior, the *left* edge(s) of objects on the input *layer* by the specified *value*, which is a non-negative floating-point number in user units. Refer to the description for how the tool determines which edge is considered a *left* edge

- BOTTOM BY *value*

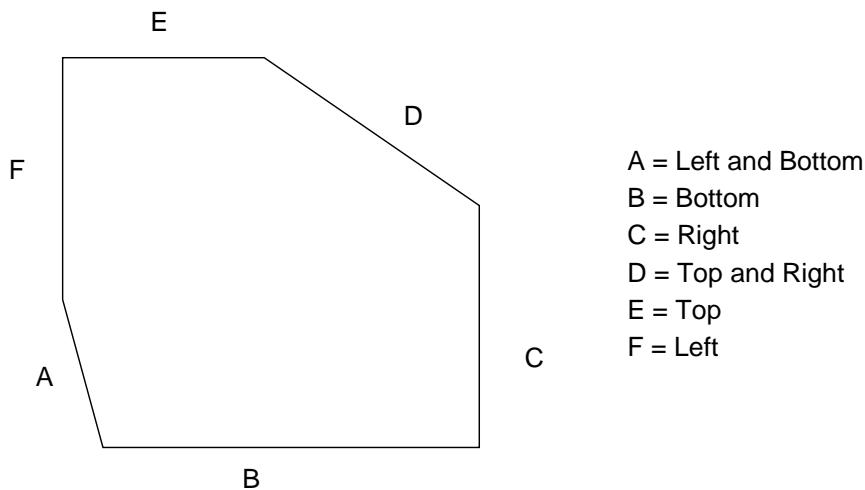
Keyword set that instructs the tool to contract, toward the interior, the *bottom* edge(s) of objects on the input *layer* by the specified *value*, which is a non-negative floating-point number in user units. Refer to the description for how the tool determines which edge is considered a *bottom* edge.

## Description

Specifies inside contraction of objects on the input layer in the direction of the x-axis, y-axis, or both.

To determine which edges are contracted, the tool uses the following:

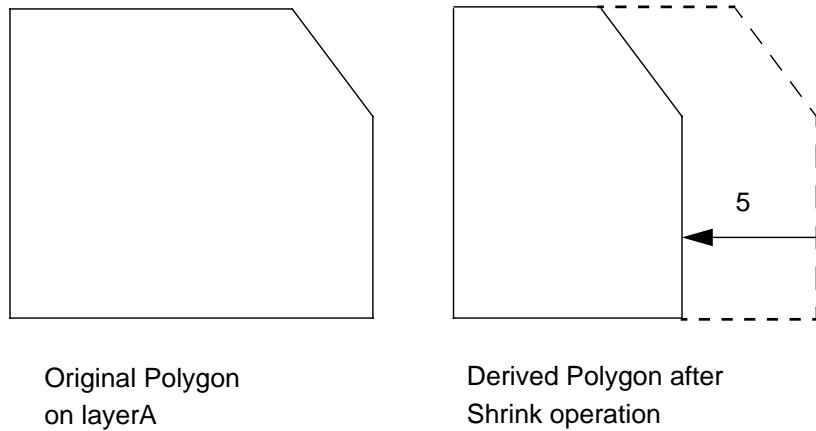
All orthogonal edges (with respect to the database axes) on the input layer are classified as right, top, left, or bottom edges dependent on how the *outside* of the edge faces. While non-orthogonal edges receive two of the four above classifications. Figure 4-319 shows how six sides of a polygon are classified by the Shrink operation.

**Figure 4-319. Edge Classification of Shrink Operation**

After the edges on the input layer have been classified, the tool contracts the edges toward the inside based on their classification and which keyword sets you specified in the rule file statement. The amount of contraction is taken from the value parameter assigned to each keyword set. Figure 4-320 shows a before and after example of a Shrink operation using the RIGHT BY keyword set.

**Figure 4-320. Shrink**

Shrink layerA RIGHT BY 5



When *layer* is a derived edge layer, all of the resulting contraction is merged and output. Otherwise, all of the resulting contraction is subtracted from the input layer and the remainder is output.

For hierarchical applications, degradation of the output layer and performance of the operation can occur due to cells that do not have consistent transforms to the flat view of the design and as a function of the parameters to the operation. Geometry must be recursively promoted until consistency can be achieved. The objects can then be shrunk in the cell where the consistency is

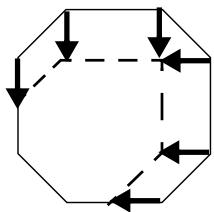
achieved. This insures that results are correct from the flat view of the design. For example, the rule check:

```
rule { SHRINK layerX RIGHT BY 1 LEFT BY 1 }
```

cannot be done within any cell where there is a flat view transform of a placement of the cell having a 90 or 270 degree rotational component and a flat view transform which does not. For instance, if placements A and B of a cell exist in cell C, you cannot do a Shrink operation on a layer in C if placement B is rotated 90 or 270 degrees with respect to A. This is due to the directions TOP, RIGHT, BOTTOM, and LEFT being different in placements A and B.

See also [Grow](#), [Size](#), and [Expand Edge](#).

## Examples



```
shrink_A = SHRINK layerA RIGHT BY 1 TOP BY 1
// orthogonal edges classified as top or right are moved
// 1 unit in either the y- or x-direction.
// top-and-left edges are moved 1 unit in the y-direction.
// bottom-and-right edges are moved 1 unit in the x-direction.
// top-and-right edges are moved 1 unit in the x- and
// y-directions. edges that match two of the orientation
// criteria contract.
```

## Size

Layer operation

**SIZE** *layer1* **BY** *size\_value*  
[OVERLAP ONLY | OVERUNDER | UNDEROVER  
| {INSIDE OF *layer2* [STEP *step\_value*] }  
| {OUTSIDE OF *layer2* [STEP *step\_value*] }]  
[TRUNCATE *trunc\_dist*]  
[BEVEL *value*]

### Summary

Expands or shrinks polygons on the input layer by specified amounts.

### Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- **BY *size\_value***  
A required keyword set, where ***size\_value*** is a floating-point value in user units, which specifies how much to expand or shrink ***layer1*** polygons. Positive numbers expand polygons, negative numbers shrink them. This value must be an even multiple of the database precision. The ***size\_value*** must be a positive number if you include the OVERLAP ONLY, OVERUNDER, UNDEROVER, or BEVEL keywords.
- **OVERLAP ONLY**  
An optional keyword specifying that the output consists only of regions where the oversized polygons overlap (not the oversized polygons themselves). This is overlap of distinct oversized polygons. The ***size\_value*** must be a positive number if you use the OVERLAP ONLY keyword. In nmLVS-H, specifying OVERLAP ONLY can cause hierarchical degradation; it is not recommended for use with hcells. This keyword cannot be specified with the OVERUNDER, UNDEROVER, INSIDE OF, OUTSIDE OF, or BEVEL keywords.
- **OVERUNDER**  
An optional keyword that instructs the tool to perform two consecutive Size operations. The ***layer1*** is first increased in size, then decreased in size, based on ***size\_value***, where ***size\_value*** must be a positive number. This keyword cannot be specified with the UNDEROVER, OVERLAP ONLY, INSIDE OF, OUTSIDE OF, or BEVEL keywords.  
  
For applications involving finding wide metal polygons, or similar types of applications where width is a concern, consider using the ([Not](#)) [With Width](#) operations.

- UNDEROVER

An optional keyword that instructs the tool to perform two consecutive Size operations. The *layer1* is first decreased in size, then increased in size, based on *size\_value*, where *size\_value* must be a positive number. This keyword cannot be specified with the OVERUNDER, OVERLAP ONLY, INSIDE OF, OUTSIDE OF, or BEVEL keywords.

For applications involving finding wide metal polygons, or similar types of applications where width is a concern, consider using the (Not) With Width operations.

- INSIDE OF *layer2*

Optional keyword followed by an original layer or layer set, or a derived polygon layer, which causes *layer1* to expand inside of *layer2*. See the [INSIDE OF and OUTSIDE OF Keywords](#) for a detailed description. This keyword cannot be specified with the OVERLAP ONLY, OVERUNDER, UNDEROVER, or OUTSIDE OF keywords.

- OUTSIDE OF *layer2*

Optional keyword followed by an original layer or layer set, or a derived polygon layer, which causes *layer1* to expand outside of *layer2*. This keyword cannot be specified with the OVERLAP ONLY, OVERUNDER, UNDEROVER, or INSIDE OF keywords.

- STEP *step\_value*

An optional keyword and positive real number that specify the step size of the sizing process. This keyword causes the sizing to occur incrementally in steps of the *step\_value*. The *step\_value* must be evenly divisible by the database precision. This keyword must be specified with either INSIDE OF or OUTSIDE OF. See the [INSIDE OF and OUTSIDE OF Keywords](#) for recommended values of *step\_value*.

- TRUNCATE *trunc\_dist*

An optional keyword set that controls spike truncation, where *trunc\_dist* is a non-negative floating-point value in absolute user units. See [TRUNCATE Keyword](#) for details of both the default truncation process and the user defined truncation process.

- BEVEL *value*

An optional keyword and positive integer that allows orthogonal corners (two edges with an interior angle of 90 degrees) on *layer1* to be approximated by *value* segmented arcs on the output layer. A BEVEL *value* of 4 is normally sufficient, and 16 should be considered a practical maximum. See [BEVEL Keyword](#) for more details. BEVEL may not be specified with OVERLAP ONLY, OVERUNDER, or UNDEROVER.

## Description

Expands or shrinks all *layer1* polygons by *size\_value*. A positive *size\_value* expands *layer1* polygon edges outward by that amount. A negative *size\_value* shrinks *layer1* polygon edges inward by the absolute value of that amount. When *size\_value* equals zero, then the output layer is a copy of *layer1*.

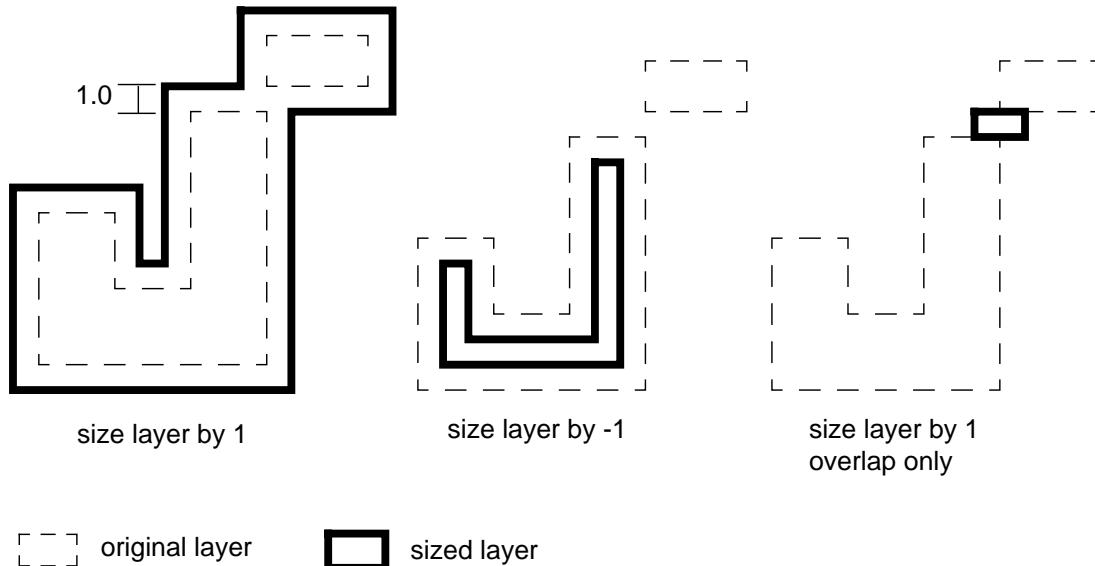
This is a node-preserving operation if you specify *size\_value* as less than or equal to zero.

## Size

Size can be used in some cases to reduce the layer data that other layer operations have to manipulate. This is done to increase rule file efficiency. See “[Efficient Width and Spacing Checks](#)” in the *Calibre Verification User’s Manual*.

Figure 4-321 shows operations that oversize and undersize the layer1 polygon by one user unit.

**Figure 4-321. Oversizing and Undersizing a Polygon**



**Requirements and Restrictions** — Using Size to derive polygons of specified width can lead to excessively complicated layer derivations, especially where angled geometry is involved. Size is not designed as a generic polygon width measurement operation. The ([Not](#)) [With Width](#) operations are more efficient than Size at finding polygons of specified widths and should be used instead of Size where width measurements are needed.

See also [Grow](#), [Shrink](#), and [Expand Edge](#).

## INSIDE OF and OUTSIDE OF Keywords

The INSIDE OF and OUTSIDE OF keywords allow checking of latch-up rules without having to enter long sequences of Size and AND (or NOT) operations. Both *size\_value* and *step\_value* parameters must be positive if INSIDE OF or OUTSIDE OF is specified.

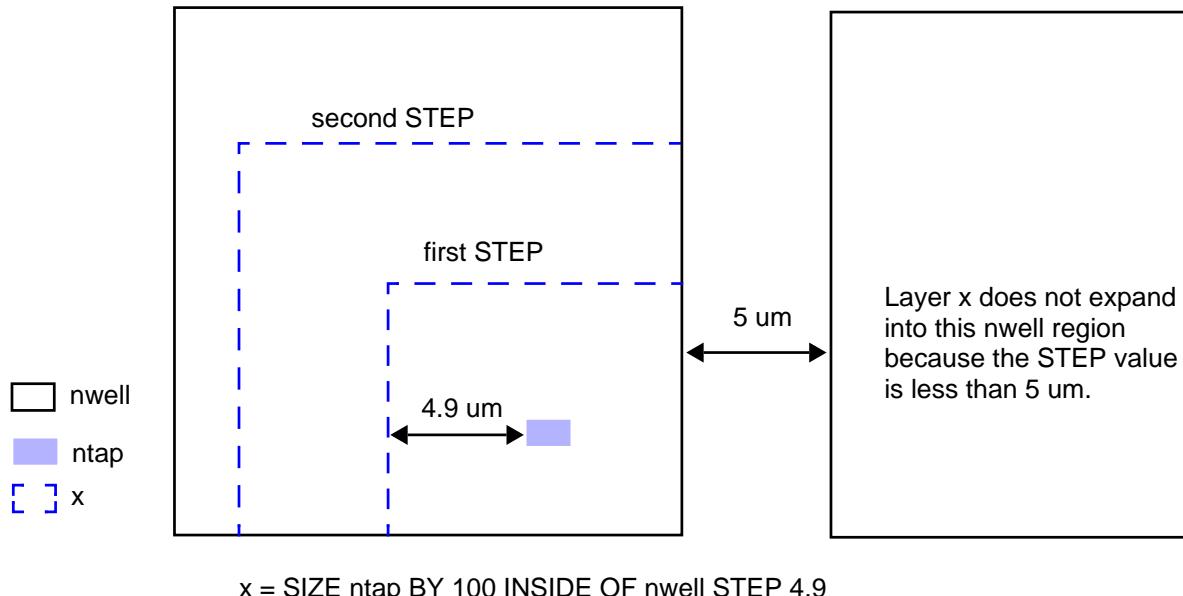
Here is an example:

```
bad_pdevice {
    // Worm rule to check tap-to-pgate spacing inside of enclosing
    // nwell polygons.
    // Gates farther than 100 um from a tap inside the same nwell
    // polygon are bad.
    // The STEP increment should be marginally less than minimum nwell
    // spacing, which is 5 um for this process.
    // This keeps the sizing inside of the enclosing nwell polygons.
```

```
x = SIZE ntap BY 100 INSIDE OF nwell STEP 4.9
pgate NOT x
}
```

[Figure 4-322](#) shows how the layer x expands through its first two STEP increments. For larger nwell polygons, the expansion would continue out to 100 um.

**Figure 4-322. Worm Rule Check**



$x = \text{SIZE ntap BY 100 INSIDE OF nwell STEP } 4.9$

The *step\_value* is normally chosen as the smaller of the notch or spacing design rules for the second input layer in the INSIDE OF case, and as the width rule in the OUTSIDE OF case. In general, a *step\_value* larger than the minimum spacing for the second input layer is not recommended.

If you do not choose a *step\_value* that is smaller than the minimum spacing for the second input layer in the INSIDE OF case, then flat and hierarchical run results may differ. This is due to multiple optimizations within the operation that favor performance improvement over consistency of results presentation based on run mode.

As shown in [Figure 4-322](#), the algorithm for bounded size causes the first input layer to crawl through the inside of the second input layer (INSIDE OF *layer2*). More precisely, assume this:

$Z = \text{SIZE A BY V INSIDE OF B [ STEP S ]}$

The algorithm proceeds as follows. Explanatory comments are interspersed with the steps. The values from the bad\_pdevice rule check example shown previously appear in the comments.

1. If STEP is omitted or if  $S > V$ , then set  $S = V$

In bad\_pdevice, STEP is specified,  $S = 4.9$ , and  $V = 100$ .

2. Let  $D = \text{FLOOR}(V / S)$  and  $R = V - (D * S)$

For the example,  $D = 20$  and  $R = 2$ .  $D$  is an integer quotient.  $R$  is a remainder value.

3. Initialize  $\text{TEMP} = A$

$\text{TEMP} = \text{ntap}$

4. Do steps 5 and 6 recursively  $D$  times

Steps 5 and 6 are done recursively 20 times for the example.

5.  $\text{TEMP1} = \text{SIZE TEMP BY } S$

The  $\text{TEMP}$  layer is sized by 4.9. For the first iteration,  $\text{TEMP}$  is the  $\text{ntap}$  layer. For subsequent iterations,  $\text{TEMP}$  is an expanded  $\text{ntap}$  layer.

6.  $\text{TEMP} = \text{TEMP1 AND B}$

The  $\text{TEMP}$  layer is the logical intersection of  $\text{TEMP1}$  and  $\text{nwell}$ . When the step size is marginally less than minimum  $\text{nwell}$  spacing, this keeps  $\text{TEMP1}$  inside the enclosing  $\text{nwell}$  polygon.

7. If  $R$  is 0, then go to step 10.

$R = 2$  for the example, so continue.

8.  $\text{TEMP} = \text{SIZE TEMP BY } R$

Expand  $\text{TEMP}$  by 2.

9.  $\text{TEMP} = \text{TEMP AND B}$

$\text{TEMP}$  is the intersection of  $\text{TEMP}$  and  $\text{nwell}$ .

10. Set  $Z = \text{TEMP}$

The output layer is the final geometry of  $\text{TEMP}$ .

The corresponding algorithm for **OUTSIDE OF** replaces AND by NOT in Steps 6 and 9.

## TRUNCATE Keyword

The **Size** operation can result in spike formation in the output polygon. The **Size** operation includes a truncation process by default, or you can control the truncation process with the **TRUNCATE** *trunc\_dist* keyword and argument. The formation of spikes and the default truncation process are described first.

The **Size** operation constructs expanded or contracted polygons with edges parallel to the original polygons. If two edges of the *layer1* polygon intersect at an angle smaller than 45 degrees (called a spike), and the **Size** operation pulls the intersecting edges in the direction of the spike when it expands or shrinks the *layer1* polygon, then the resulting spike is truncated to a distance of  $1/\cos 67.5$  (approximately 2.61) times the absolute value of the *size\_value*. The

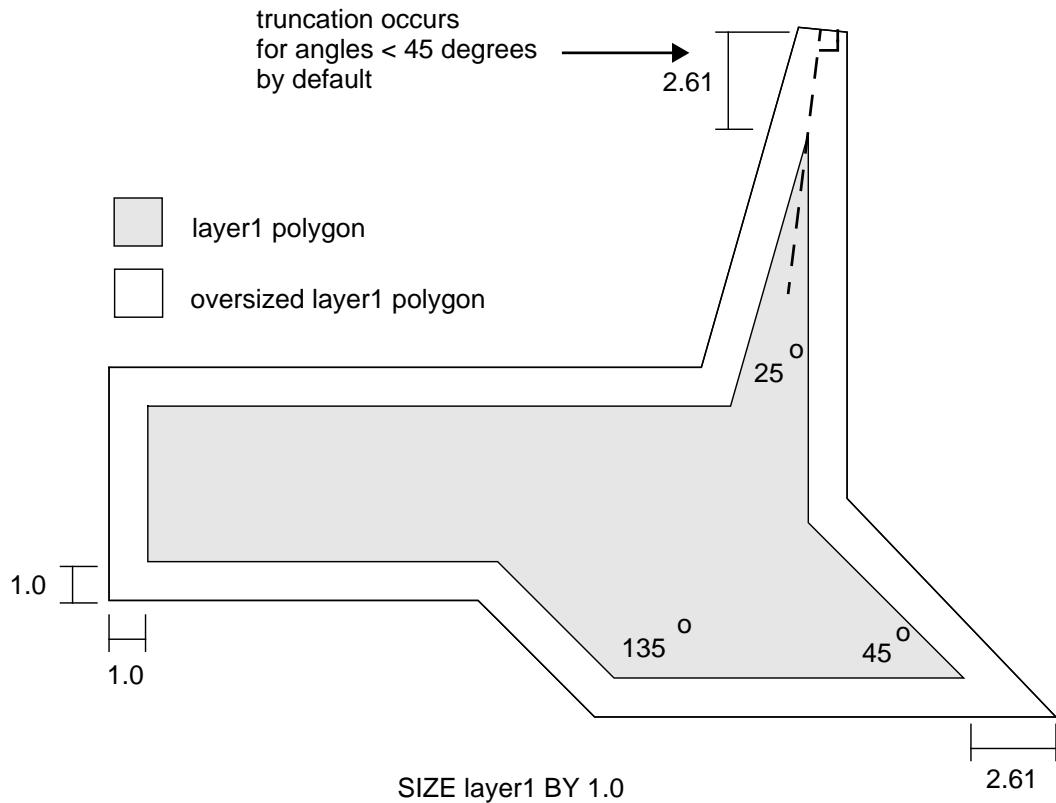
absolute value of *size\_value* multiplied by  $1/\cos 67.5$  is the distance that a 45-degree corner would be moved by the Size operation along the direction—toward or away from—a spike.

If the *trunc\_dist* is greater than or equal to the absolute value of *size\_value*, which is true by default, then the behavior in the following discussion applies.

The truncation consists of constructing a new edge perpendicular to the bisector of the angle between the original intersecting edges. However, if one of the polygon edges contributing to the spike is orthogonal to the database axes, then the truncating segment will be placed perpendicular to the orthogonal edge, rather than perpendicular to the bisector of the spike. In this case, the truncation distance is measured along the orthogonal edge.

Figure 4-323 shows the default truncation process with the truncating segment placed at the bisector of the spike. In the figure the 25 degree spike is truncated, but the 45 degree spike is not. The default truncation distance of  $1/\cos 67.5$  times *size\_value* means that only angles of less than 45 degrees ever result in spike truncation.

**Figure 4-323. Handling of Polygon Spike**



You can specify the truncation distance using the TRUNCATE *trunc\_dist* keyword and value, where *trunc\_dist* is a non-negative floating point number in absolute user units that specifies the truncation distance. For example, *trunc\_dist* values smaller than the default (2.61 times *size\_value*) can cause truncation of acute angles greater than 45 degrees. When the TRUNCATE keyword is used, *trunc\_dist* is not used as a multiplier of *size\_value*.

You can use the TRUNCATE keyword set to inhibit excessive spike growth when attempting to find areas of well-behaved geometry (only orthogonal and 45-degree edges) by using undersizing followed by oversizing. In some instances, the undersize can produce 45-degree angles which grow excessively when oversized. Setting *trunc\_dist* to the absolute value of *size\_value* may limit the spike growth and provide more accurate results. This is demonstrated in the last figure of [Example 2](#).

If the *trunc\_dist* is less than the absolute value of *size\_value*, then the behavior described previously is modified as in the following discussion.

---

**Note**

It is not recommended to use a *trunc\_dist* that is less than the absolute value of *size\_value*.

---

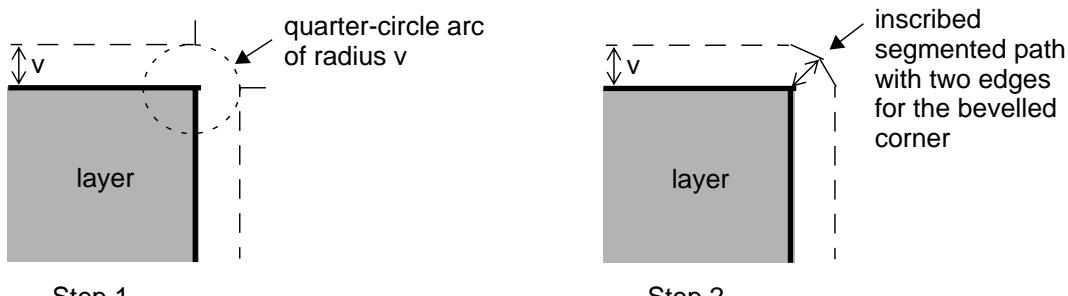
If an interior angle exceeds 60 degrees (for oversizing), or is less than 300 degrees (for undersizing), and one edge at the angle is orthogonal to a database axis, then truncation will be perpendicular to the bisector of the angle, as in the general case. This is true even if one of the edges forming the angle is orthogonal to the database axes.

The truncation distance, *trunc\_dist*, may be adjusted upward toward the absolute value of *size\_value* as a function of the interior angle between the edges. In most cases, changes will only be apparent if the interior angle is close 90 degrees for oversizing and 270 degrees for undersizing.

## BEVEL Keyword

BEVEL is primarily useful for high-accuracy worm rule checking in conjunction with the INSIDE OF *layer2* option. If BEVEL is used, then a quarter-circle arc with a radius of the *size\_value* and with a center at the vertex of the perpendicular edges is constructed first, internally. The actual output edges are then formed by inscribing a segmented path in the arc. The edge count of the segmented path is equal to the BEVEL *value*. The endpoints of the edges that form this path are spaced equidistant along the arc. (See [Figure 4-324](#).)

**Figure 4-324. Size BEVEL**



SIZE layer BY v BEVEL 2

The greater the BEVEL *value*, the slower the Size operation will be (especially when using INSIDE OF for small STEP values). This is because the number of constructed edges can be very large and Size is very careful with skew edge handling. A BEVEL *value* of 4 is normally sufficient, and 16 should be considered a practical maximum.

## Examples

### Example 1

This example finds vias within a minimum spacing.

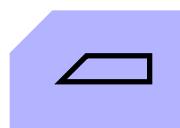
```
bad_via {
    // find overlaps of vias expanded by minimum spacing distance.
    x = SIZE via BY min_via_space OVERLAP ONLY
    // vias that cut the overlap regions are output
    via CUT x
}
```

### Example 2

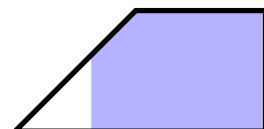
This shows some basic sizing examples using non-rectangular regions. The angled edge of the original polygon is nearly 45 degrees. In the third and fourth examples, the first Size command constructs an intermediate layer, and the black outline shows the final polygon result.



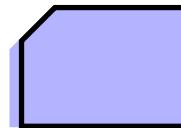
original polygon



SIZE layer BY -v



X = SIZE layer BY -v  
SIZE X BY v



X = SIZE layer BY -v  
SIZE X BY v TRUNCATE v

### Example 3

The following Size operation, using OVERUNDER:

```
new_layerA = SIZE metall1 BY 2 OVERUNDER
```

is equivalent to the following rule file statements:

```
layer_a = SIZE metall1 BY 2
new_layerA = SIZE layer_a BY -2
```

## Snap

Layer operation

**SNAP layer {*r* | *x* *y*}**

### Parameters

- *layer*  
A required original or derived polygon layer.
- *r* | *x* *y*  
Required positive integers, in database units, which specify the snap grid. The value *r* is applied in both x- and y- directions, while *x* is applied in the x-direction and *y* in the y-direction.

### Description

Snaps each vertex on the input layer to the grid specified by the resolution parameter(s). The parameters should be ordered as shown to avoid ambiguity. The grid parameters have no interaction with [Resolution](#).

Snapping always preserves 45-degree edges between two edges orthogonal to the database axes if the snap resolution has equal *x* and *y* values. For hierarchical Calibre applications, snapping to unequal *x* and *y* resolutions is not permissible and the least common multiple of the two resolutions is used instead.

See also [Offgrid](#), [Snap Offgrid](#), [DRC Boolean Nosnap45](#), [Drawn Offgrid](#), and [Flag Offgrid](#).

### Examples

#### Example 1

The following example snaps polygons on diffusion to a grid of 0.01 user units (assuming 1000 database units per user-unit):

```
snapped_diff = SNAP diff 10
```

#### Example 2

The following example snaps metal to a 2 database unit resolution grid because 3 -1 is viewed as the arithmetic operation 3 minus 1:

```
rule { SNAP metal 3 -1 }
```

whereas the following example results in a compilation error due to the negative y-value:

```
rule { SNAP metal 3 (-1) }
```

# Snap Offgrid

Specification statement

## SNAP OFFGRID {NO | YES}

### Parameters

- **NO**

Keyword that instructs the tool to keep original layer shapes in their original locations. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs the tool to snap original layer shapes to the grid specified in the [Resolution](#) or appropriate [Layer Resolution](#) specification statements. This statement also instructs the tool to snap [DRC Map Text](#) objects as specified by the Resolution specification statement. In the latter case, any Layer Resolution specification statement does not override the Resolution statement.

### Description

Specifies that, prior to merging, original layer shapes are snapped to the grid specified in the Resolution specification statement or the appropriate Layer Resolution specification statement, if present. This statement can be specified once in your rule file.

Snapping occurs prior to any acute, skew, or off-grid flagging (see “[Flagging Bad Original Shapes](#)” in the *Calibre Verification User’s Manual*) and always preserves the 45-degree edges between two edges orthogonal to the database axes if the snap resolution has equal x and y values.

For hierarchical Calibre applications, cell placements are snapped to grid first, then shapes are snapped on a per-cell basis. This prevents placements from becoming off grid in the flat view of the input layout database. The resolution for placement snapping is the least common multiple of all grid values specified in applicable Resolution or Layer Resolution specification statements. For resolutions where the x and y values are unequal, snapping is to the least common multiple of the two values.

For additional information, see “[Snapping Offgrid Objects to Grid](#)” in the *Calibre Verification User’s Manual*.

This statement applies only to the Mask mode in ICVerify applications.

See also [Offgrid](#), [Snap](#), [DRC Boolean Nosnap45](#), [Drawn Offgrid](#), and [Flag Offgrid](#).

### Examples

```
// Snap all offgrid vertices to grid.
SNAP OFFGRID YES
```

## Source Case

Specification statement

### SOURCE CASE {NO | YES}

#### Parameters

- **NO**

Keyword that instructs that the tool not to process the source netlist (SPICE only) in a case-sensitive manner. This is the default behavior if you do not include this statement in the rule file.

- **YES**

Keyword that instructs that the tool to process the source netlist (SPICE only) in a case-sensitive manner.

#### Description

Specifies whether the source netlist should be processed in a case-sensitive manner.

You should specify Source Case YES if the intent of the design is to have source netlist elements be treated as different elements if text cases differ. For example, if VDD and vdd should be treated as separate nets, then specifying YES is desirable.

This statement controls the treatment of node names, subcircuit names, model names, and user-defined parameter names such as subcircuit definitions, subcircuit calls, and in .PARAM statements. Keywords such as element names (M, C, X), dot (.) commands (as in .SUBCKT), and built-in parameter names (W, L, \$SUB) are not affected.

This statement controls case sensitivity in the SPICE reader module but not during LVS comparison. Use [LVS Compare Case](#) to control case-sensitive comparison.

See also [Layout Case](#).

#### Examples

##### Example 1

Suppose you have the following source netlist:

```
.SUBCKT block A B Y
X1 net1 B GND gnd cell1
...
.ENDS
```

If the intent of the design is to have GND and gnd be considered the same net, then you should specify Source Case NO.

If the intent of the design is to have GND and gnd be separate nets, then you should specify Source Case YES. If Source Case YES is specified, then it is frequently desirable that Layout Case YES be specified also. However, if the layout netlist uses node numbers instead of names, then Layout Case NO is appropriate.

If you need LVS comparison to be case sensitive, then LVS Compare Case YES or NAMES should be used.

## Source Path

Specification statement

### SOURCE PATH *filename*

Used only by Calibre nmLVS/nmLVS-H and Calibre xRC.

#### Parameters

- *filename*

A required filename of the source database.

The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in Chapter 2, “[Key Concepts](#)”.

#### Description

Specifies the source database filename for the design specified by the [Source System](#) statement. Can be specified once in your rule file. The filename specified should have these characteristics:

- For SPICE netlists, refer to the file.
- For Verilog netlists converted into extended SPICE form, refer to the respective output SPICE file.
- For CNET databases, refer to the CNET database directory.

**Compressed files** — Any SPICE *filename* ending in .gz or .Z is treated as a compressed file. Such files can be opened automatically by Calibre using gzip or uncompress, respectively, if these utilities are in your PATH variable. Some versions of these utilities cannot open files larger than two GB. If you cannot open compressed files larger than two GB, you may need to update to a newer version of gunzip or uncompress. These utilities may also have difficulty opening files across a network. Neither of these issues are Calibre limitations.

Compressed SPICE files can be either libraries named in a .LIB statement or included files named in a .INCLUDE statement. Due to the limitations of the technology of uncompression routines, it is not advisable to compress files that have .INCLUDE statements or .LIB statements that are not placed near the top of the file that reference, rather than define, a library. This is because the compressed file must always be read from the beginning when returning from processing the included file or the referenced library. For example, it is not advisable to compress files that look like this:

```
$ Long SPICE file:  
...  
... many lines of SPICE  
...  
$ Followed by .INCLUDE:  
.INCLUDE somefile  
$ Bad idea to compress this file.
```

It is preferable to reorder the file so the .INCLUDE statement appears near the top of the file.

You can specify source database information in Calibre Interactive - nmLVS. See “[Supplying Source Netlist Data](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [Source Primary](#).

## Examples

```
SOURCE SYSTEM SPICE
SOURCE PATH "/home/golden_netlists/chip.spi"
SOURCE PRIMARY "top"
```

## Source Primary

Specification statement

### SOURCE PRIMARY [*name*]

Used only in Calibre nmLVS/nmLVS-H and Calibre xRC.

#### Parameters

- *name*

A top-level cell or subcircuit name, which may be a singleton string variable, of the source database. This parameter is optional.

#### Description

Specifies a subcircuit or cell name for SPICE source systems for Calibre applications. This includes Verilog netlists that have been converted into extended SPICE format. (For more information about Verilog translation, see “[Verilog-to-LVS](#)” in the *Calibre Verification User’s Manual*.) Can be specified once in your rule file.

If [Source System](#) SPICE is specified, then the Source Primary statement is optional, with the following behaviors:

- If present, Source Primary identifies the top-level subcircuit name. Calibre nmLVS evaluates the netlist starting with that subcircuit to the bottom of the hierarchy. The pins of the top-level subcircuit serve as design ports. Any subcircuit in the netlist can be specified as a top-level subcircuit.
- When Source Primary is not present, nmLVS searches the netlist for element statements or subcircuit calls that are not part of any subcircuit. Those statements then constitute the top-level network.

Omitting the *name* parameter means that no Source Primary cell is specified. This is largely equivalent to omitting the statement altogether, but this form is useful for overrides involving the DRC ICSTATION YES specification statement used in control files generated by Calibre Interactive. (Note that DRC ICSTATION YES should only appear in control files generated by Calibre Interactive. It should not appear in any other rule files.)

Not used for CNET format systems.

You can specify source database information in Calibre Interactive - nmLVS. See “[Supplying Source Netlist Data](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See also [Source Path](#).

#### Examples

```
SOURCE SYSTEM SPICE
SOURCE PATH "/home/source_netlists/chip.spi"
SOURCE PRIMARY "top"
```

# Source System

Specification statement

## SOURCE SYSTEM {CNET | SPICE}

Used only in Calibre nmLVS/nmLVS-H and Calibre xRC.

### Parameters

- **CNET**

Keyword that specifies that the source is a compiled netlist (Cnet) database.

- **SPICE**

Keyword that specifies that the source is a SPICE, SPICE 2, HSPICE, or CDL netlist, or a Verilog netlist converted into extended SPICE form.

### Description

Specifies the source database type for Calibre applications. This statement is required for Calibre applications that execute nmLVS and can be specified once in your rule file. Must be used with a [Source Path](#) statement.

You can specify source database information in Calibre Interactive - nmLVS. See “[Supplying Source Netlist Data](#)” in the *Calibre Interactive and Calibre RVE User’s Manual*.

See “[SPICE Format](#)” in the *Calibre Verification User’s Manual* for more information about how Calibre interprets SPICE netlists.

See also [Source Primary](#).

### Examples

```
SOURCE SYSTEM SPICE
SOURCE PATH "/home/source_netlists/chip.spi"
SOURCE PRIMARY "top"
```

# Stamp

Layer operation

**STAMP *layer1* BY *layer2* [ABUT ALSO]**

## Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- **BY *layer2***  
A required keyword set, where *layer2* is an original layer or layer set, or a derived polygon layer. The connectivity on *layer2* must be established in order to use Stamp.
- **ABUT ALSO**  
An optional keyword that specifies abutment is considered to constitute overlap in this operation.

## Description

### Note



For most applications involving soft connection (or nwell jumper) checks, using [Sconnect](#) is preferable to using Stamp. There are some specialized nmDRC applications where Stamp may be useful. For xRC and xL usage, Stamp provides insufficient physical information causing layers to appear unconnected.

Stamp selects *layer1* polygons that are overlapped by *layer2* polygons, and which can receive valid connectivity information from *layer2* polygons. Warning messages report *layer1* polygons that receive missing or conflicting connectivity information. Note that the connectivity of *layer2* polygons *is not* actually passed to *layer1*; but the presence of valid connectivity information from *layer2* is tested. For example:

```
CONNECT metall welltie
rule { STAMP well BY welltie }
```

This selects all well polygons that can receive unambiguous connectivity information from wellties. For any well polygons that cannot receive valid connectivity from welltie polygons, a warning is issued in the transcript.

The most common use of the Stamp operation is to create derived polygon layers where the derived layer is essentially a copy of *layer1* polygons having *layer2* connectivity passed from *layer2* polygons. Building on the previous example:

```
CONNECT metall welltie
rule {
  x = STAMP well BY welltie /* copies well polygons having valid
  connectivity from welltie */
  well NOT x
}
```

This code selects all well polygons that are either floating or have soft connections through them caused by welltie polygons of differing nodes. (There are more efficient ways of doing this sort of check using Sconnect or Not Interact BY NET, but this example demonstrates a possible use.)

When used for layer derivation, the Stamp operation never establishes connectivity on *layer1* itself. Rather, a copy of selected *layer1* polygons is created with the connectivity information from *layer2*. This behavior differs from Dracula where the first input layer is stamped in place and it represents the stamped input layer throughout the Dracula rule file.

Use the ABUT ALSO keyword if you also want polygon abutment (touching at an edge) to constitute a valid connection.

See also [Connect](#).

## Examples

### Example 1

Suppose you want to do a spacing check on narrow metal wires that are not electrically connected. Assuming a correct derivation of narrow\_metal and that the connectivity of layer metal is established, consider:

```
narrow_metal_check {
    connected_narrow_metal = STAMP narrow_metal BY metal
    EXTERNAL connected_narrow_metal < 4 not connected }
```

It might appear that this Connect operation:

```
CONNECT narrow_metal metal
```

would have worked just as well as the Stamp operation. In this particular example, if you assume the stamped layer (narrow\_metal) is a strict subset of the layer carrying the connectivity information (metal), the effect would be equivalent to Stamp. However, there are at least two reasons why Connect operations generally should not be substituted for Stamp operations:

- The Stamp operation, being a layer operation, is executed only when required, whereas connectivity extraction is done at the beginning of a Calibre run. Using the Connect operation carelessly is costly in terms of memory consumption and calculation time. The Stamp operation results remain in memory only as long as they are needed. Connect information persists much longer in memory.
- In the general case, the Stamp operation is not stamping a strict subset of geometry from the layer possessing connectivity, as it is in this example. Connectivity established using Connect statements should be done carefully. Carelessly using Connect instead of Stamp can corrupt circuit connectivity.

**Example 2**

There are other methods than the previous example to establish connectivity. Because the node-preserving operations pass node numbers from their first input layer to a derived layer, one way to pass connectivity is as follows:

```
connect metal
NM1 = SIZE metal BY -1.99
// Metal with width < 4 goes away. SIZE is node-preserving.
NM2 = SIZE NM1 BY 1.99
// Metal having width greater than or equal to 4 restored.
narrow_metal = metal NOT NM2
/* Connectivity is passed from metal to narrow_metal by the
NOT operation. */
```

or, to get edges on narrow metal:

```
NM1 = SIZE metal BY -1.99 // Metal < 4 goes away.
NM2 = SIZE NM1 BY 1.99    // Metal >= 4 restored.
narrow_metal = metal NOT COINCIDENT EDGE NM2
/* Narrow metal edges with connectivity passed from the
NOT COINCIDENT EDGE operation. */
```

## SVRF Error

Specification statement

### SVRF ERROR “*message*”

#### Parameters

- *message*

A string that is typically a message to be displayed as a compilation error.

#### Description

This specification statement causes a compiler error containing the given *message*. This means the SVRF Error statement should appear within a rule file conditional statement. See “[Pre-Processor Directives](#)” on page 56.

The message produced has the following form:

```
ERROR: Error USER1 on line <line-number> of <file-name> - <message text>
```

where <message text> is the string parameter of the SVRF Error command. The error message appears in the STANDARD VERIFICATION RULE FILE COMPILATION MODULE portion of the run transcript.

See also [SVRF Message](#) for cases in which you do not want the run to halt with a compilation error.

#### Example

```
#ifndef METAL6
SVRF ERROR "Do not run these rules unless METAL6 is set"
#endif
```

## SVRF Message

Specification statement

### SVRF MESSAGE “*message*”

#### Parameters

- “*message*”

A quoted string that is printed in the run transcript at compilation time.

#### Description

This specification statement prints the given *message* in the run transcript. This statement may be specified any number of times. The message printed has the following form:

```
SVRF MESSAGE on line <line-number> of <file-name> - <message text>
```

where <message text> is the string parameter of the SVRF Message statement. For example:

```
SVRF MESSAGE on line 5 of rule_file - "Calibre nmDRC is fun to use."
```

It appears in the STANDARD VERIFICATION RULE FILE COMPILATION MODULE portion of the run transcript.

See also [SVRF Error](#) for cases in which you want the run to halt with a compilation error.

#### Examples

```
// echo this message during the run transcript
SVRF MESSAGE "Starting metal width checks"
```

## SVRF Version

Specification statement

### SVRF VERSION “*version\_id*”

#### Parameters

- “*version\_id*”

A quoted string that identifies a Calibre version. The string is of this form:

vDDDD.D\_D\*.D\*

where D is any decimal digit, D\* is one or more digits, and the other characters are literals.

#### Description

Specifies that the Calibre version used for the run matches the *version\_id*, or is a later version. If an earlier version than the *version\_id* is used, then a compiler error results.

This statement may be specified multiple times, in which case the latest version specified is the earliest version that can be used.

Specifying Calibre versions earlier than 2008.3\_16.12 is not useful because this statement did not exist prior to this version, and it causes a compiler error in earlier versions.

#### Examples

```
// use this Calibre version, or later
SVRF VERSION "v2008.3_16.12"
```

## TDDRC

Layer operation

**TDDRC** *layer* [REGION] [BY POLYGON [MEASURE ALL]]  
  {  
    *measurement\_rule* [OPPOSITE | OPPOSITE EXTENDED *value* | SQUARE]  
    [PROJECTING *constraint*] }  
  [*measurement\_rule* ...]

Used only by Calibre.

### Summary

Layer operation for performing relatively simple table-based DRC checks. Except for automatically combining output from numerous individual operations, TDDRC may be approximately modeled by equivalent SVRF code. The intent is to provide a performance advantage over the equivalent SVRF code, as well as ease of entry into the rule file.

### Parameters

- ***layer***

A required original or derived polygon layer. If specified by itself, with no brackets surrounding the layer name, this produces error-directed output, which cannot be used for derived layer input to any other operation (except certain DFM operations). For edge-directed output (which can be used for deriving layers), enclose the ***layer*** parameter in brackets [ ]. This is similar to edge-directed output for the [External](#) and [Internal](#) operations.

- **REGION**

An optional keyword that generates a derived polygon layer instead of a derived error layer. (This output can be used in layer derivations.) It constructs edge projections between the endpoints of selected edges to create polygonal regions; the composite of the selected edges and edge projections is output as derived polygon data. This is similar to polygon-directed output for the EXTERNAL and INTERNAL operations. This keyword cannot be used simultaneously with the brackets [ ] used to generate derived edge layers.

If you are deriving polygon layers, using the REGION keyword with the Euclidean (default) metric can generate large numbers of skew edges, which must be flattened in hierarchical applications. This is usually wasteful and requires much excess processing time. You can use the OPPOSITE metric to eliminate skew edges, but be aware that you can miss legitimate design rule errors (such as corner-to-corner configurations) when using OPPOSITE.

- **BY POLYGON**

An optional keyword that specifies for certain ***measurement\_rule*** types to alter internal heuristics to perform measurements between derived polygons rather than derived edges.

- **MEASURE ALL**

An optional keyword that specifies to ignore the polygon containment criteria for certain measurements when BY POLYGON is specified. This allows EXternal dimensional check operations to “see through” polygons that ordinarily they would not.

- ***measurement\_rule***

A mandatory rule that specifies the measurements to perform. There must be at least one ***measurement\_rule*** in a rule check. There are two basic types of measurement rules—spacing and width. Both spacing and width rules may be specified in the same rule check. The measurement rules have these basic forms:

- **SPACE** *space\_constraint* [WIDTH1 *constraint1*] [WIDTH2 *constraint2*]  
[EXCLUDE SHIELDED [*level*]]

**SPACE** *space\_constraint* — Optional keyword and constraint that measure spacing between edges on the input layer. When specified alone, the SPACE parameter set measures all edges on the input ***layer*** to see if they meet the *space\_constraint*. When specified with either WIDTH1 or WIDTH2, or both, then the width of polygons is taken into account before edge spacing measurements are made.

**WIDTH1** *constraint1* — Optional keyword and constraint that must be specified with **SPACE** *space\_constraint*. When this is specified as the only width condition in a spacing rule, edge spacings are tested where *all* of the edges being measured come from polygons having a width that meets *constraint1*. When specified with **WIDTH2**, edge spacings are tested between edges coming from polygons having widths that meet *constraint1* and *constraint2*, respectively.

**WIDTH2** *constraint2* — Optional keyword and constraint that must be specified with **SPACE** *space\_constraint*. When this is specified as the only width condition in a spacing rule, edge spacings are tested where *one* of the edges being measured comes from a polygon having a width that meets *constraint2*. The other edge comes from any polygon on the input layer that lies within the measurement region. When specified with **WIDTH1**, edge spacings are tested between edges coming from polygons having widths that meet *constraint1* and *constraint2*, respectively.

**EXCLUDE SHIELDED** [*level*] — An optional keyword and integer parameter that specify to suppress the edge measurement process to various levels due to the presence of shielding edges. A *shielding edge* is an edge S, which completely or partially blocks the line-of-sight between edges A and B. This process is described in detail in the [Edge Shielding](#) section.

The optional *level* specifies the amount of effort the tool expends to identify a shielding edge. The *level* is an integer from 0 to 4, with 4 being used if no *level* is specified with **EXCLUDE SHIELDED**. Specifying 0 indicates no effort is expended to do this, with increasing levels of effort through level 4. Level 4 provides maximum shielding effect, and also requires the greatest amount of processing time. If **EXCLUDE SHIELDED** is not specified, level 1 is used. Level 0 is used if **MEASURE ALL** is specified.

- WIDTH *width\_constraint*

Optional keyword and constraint that measure the widths of polygons on the input *layer*. This is different from the WIDTH1 or WIDTH2 keywords in that WIDTH rules are distinct from SPACE rules. All edges on the input layer are tested to see if they meet the *width\_constraint*.
- OPPOSITE | OPPOSITE EXTENDED *value* | SQUARE

Optional keywords that specify the measurement metric to use in a spacing or width rule, described previously. Only one of the three metrics may be specified in a spacing or width rule. The default metric is Euclidean, and there is no keyword needed for this metric. These metrics are discussed under EXTERNAL and INTERNAL, and under “Metrics” in the *Calibre Verification User’s Manual*.
- PROJECTING *constraint*

Optional keyword set used in a spacing or width rule, which instructs the rule to measure the separation between edges based upon their mutual edge projection. Note that the projections of edges onto other edges (if any) are not the output, rather the results of the **measurement\_rule** and associated keywords are output if the specified projection exists. Specifying the PROJECTING keyword also internally sets the PARALLEL ONLY filter for measurements. Edge projection and the PARALLEL ONLY filter are discussed under EXTERNAL and INTERNAL.

## Description

TDDRC (Table-Driven DRC) allows a subset of the one-layer EXTERNAL and INTERNAL operations’ functionality to be checked in a table-based manner. The operation is a non-node-preserving layer constructor.

Here is a basic example with error-directed output:

```
M1 { @ Minimum spacing for M1 is 0.01. Minimum width for M1 is 0.01.  
@ Minimum spacing for M1 of width >= 0.01 < 0.04 is 0.02.  
@ Minimum spacing for M1 of width >= 0.04 < 0.06 is 0.04.  
@ Minimum spacing for M1 of width >= .06 < 0.08 and M1 of width == 0.08  
@ is 0.06  
  
    TDDRC M1  
    SPACE < 0.01  
    WIDTH < 0.01  
    SPACE < 0.02 WIDTH1 >= 0.01 < 0.04  
    SPACE < 0.04 WIDTH1 >= 0.04 < 0.06  
    SPACE < 0.06 WIDTH1 >= 0.06 < 0.08 WIDTH2 == 0.08  
}
```

If you want edge-directed output for the previous example, use this syntax with the literal brackets []:

```
x = TDDRC [M1] ...
```

TDDRC is fully supported by MT and MTflex processes. No special licenses are required.

## Definitions of Measurement Rules

These are the formal definitions of all possible width and spacing rules. All of these types of measurement rules can appear in any TDDRC rule check. For each rule shown, the *metric* refers to the OPPOSITE, OPPOSITE EXTENDED *value*, or SQUARE keywords.

- SPACE *space\_constraint [metric] [PROJECTING constraint]*

is defined as:

EXT *layer space\_constraint [metric]*  
[PROJECTING constraint PARALLEL ONLY] ABUT < 90 SINGULAR

- SPACE *space\_constraint [metric] [PROJECTING constraint] WIDTH1 constraint1*

is defined as:

*x = layer COINCIDENT EDGE (layer WITH WIDTH constraint1)*  
EXT *x space\_constraint [metric] [PROJECTING constraint PARALLEL ONLY]*  
ABUT < 90

If BY POLYGON [MEASURE ALL] is specified, then the following operations are used:

*x = layer WITH WIDTH constraint1*  
EXT *x space\_constraint [metric] [PROJECTING constraint PARALLEL ONLY]*  
ABUT < 90 SINGULAR

- SPACE *space\_constraint [metric] [PROJECTING constraint] WIDTH2 constraint2*

is defined as:

*x = layer COINCIDENT EDGE (layer WITH WIDTH constraint2)*  
EXT *x layer space\_constraint [metric]*  
[PROJECTING constraint PARALLEL ONLY] ABUT < 90

If BY POLYGON [MEASURE ALL] is specified, then the following operations are used:

*x = layer WITH WIDTH constraint2*  
EXT *x space\_constraint [metric] [PROJECTING constraint PARALLEL ONLY]*  
ABUT < 90 SINGULAR [MEASURE ALL]

- SPACE *space\_constraint [metric] [PROJECTING constraint]*  
WIDTH1 *constraint1* WIDTH2 *constraint2*

is defined as:

*x = layer COINCIDENT EDGE (layer WITH WIDTH constraint1)*  
*y = layer COINCIDENT EDGE (layer WITH WIDTH constraint2)*  
EXT *x y space\_constraint [metric] [PROJECTING constraint PARALLEL ONLY]*  
ABUT < 90

If BY POLYGON [MEASURE ALL] is specified, then the following operations are used:

```
x = layer WITH WIDTH constraint1
y = layer WITH WIDTH constraint2
EXT x y space_constraint [metric] [PROJECTING constraint PARALLEL ONLY]
    ABUT < 90 SINGULAR [MEASURE ALL]
```

- **WIDTH width\_constraint [metric] [PROJECTING constraint]**

is defined as:

```
INT layer width_constraint [metric]
    [PROJECTING constraint PARALLEL ONLY] ABUT < 90 SINGULAR
```

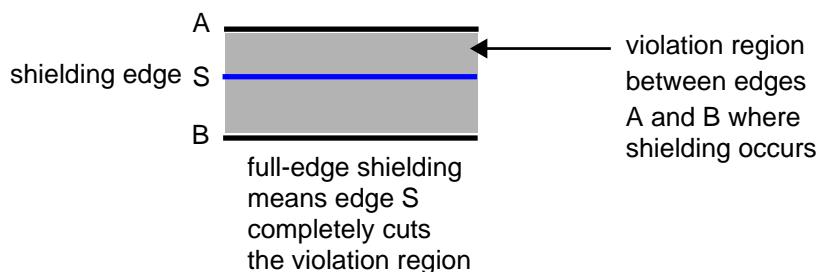
## Edge Shielding

When two edges A and B are measured by TDDRC, output from the measurement process can be suppressed to various extents due to the presence of shielding edges. A *shielding edge* is an edge S that completely or partially blocks the line-of-sight between edges A and B. Edge shielding is controlled in a SPACE rule by using this keyword set:

```
EXCLUDE SHIELDED [ 0 | 1 | 2 | 3 | 4 ]
```

[Figure 4-325](#) shows how shielding occurs.

**Figure 4-325. Full-edge Shielding**



The integers 0, 1, 2, 3, or 4 may be specified as the *level*, depending upon the level of shielding required. Integers greater than 4 are considered equivalent to 4, which is also the default if EXCLUDE SHIELDED is specified without an integer.

Level 0 means that no shielding is performed (which is less than what the tool does by default when EXCLUDE SHIELDED is not specified). Level 0 is used if MEASURE ALL is specified.

Levels 1, 2, and 3 all perform full-edge shielding. *Full-edge shielding* suppresses output from the measurement of edges A and B by the presence of a shielding edge S, which completely cuts the region defining the violation (in the sense of that produced by the REGION keyword of the EXternal operation) into two distinct areas. The only distinction between levels 1, 2, and 3 is the amount of processing time used to locate the shielding edges.

Level 1 requires a shielding edge S to intersect an endpoint of edge A or B. See the section “False Measurement Reduction” in the *Calibre Verification User’s Manual* for a discussion of this type of shielding. Level 1 is used when EXCLUDE SHIELDED and MEASURE ALL are not specified.

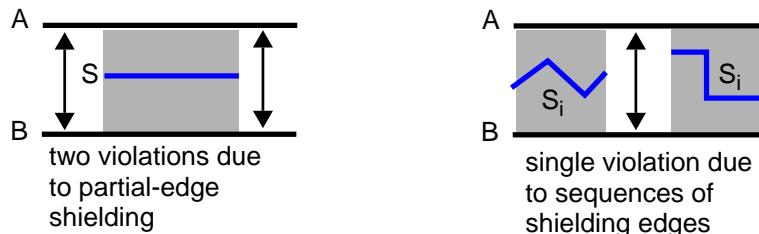
Level 2 is primarily for Mentor Graphics research and development use and is not discussed further.

Level 3 guarantees that any shielding edge S is located, but this comes at the cost of substantially increased runtime.

Level 4, like level 3, guarantees the location of any shielding edge S, and performs *partial-edge shielding* (sometimes referred to as *shadowing*). Partial-edge shielding does not require S to cut the violation region between edges A and B completely (if it does, then full-edge shielding applies). Rather, only the actual portions of the violations between A and B, which have their line-of-sight blocked by S, are suppressed.

Partial-edge shielding can produce multiple violations between edges A and B where no shielding would produce one violation. Partial-edge shielding also completely suppresses the violation between edge A and B if a sequence of shielding edges  $S_i$  collectively cut the violation region. Partial-edge shielding is illustrated in [Figure 4-326](#).

**Figure 4-326. Partial-Edge Shielding**



EXCLUDE SHIELDED is ignored in any of the following cases:

- A and B intersect.
- the operation is a two-layer operation with a closed interval measurement constraint and the default or SQUARE metric is specified.
- the operation is a two-layer operation with NOT PROJECTING specified.

For performance reasons, the OPPOSITE metric is highly recommended when using levels 3 and 4.

Special care to find all shielded edges, akin to what is applied with the EXCLUDE FALSE keyword for the dimensional check operations, is exercised (for a performance cost) when EXCLUDE SHIELDED is specified with a value greater than 2.

## Examples

### Example 1

The following example is a table-based rule check:

```
m1_spacings_and_widths {
    TDDRC metal1
    SPACE < 0.5 WIDTH1 >= 0.10
    SPACE < 0.4 WIDTH1 >= 0.05 < 0.10 WIDTH2 >= 0.05 < 0.10
    SPACE < 0.3 WIDTH1 < 0.05           WIDTH2 >= 0.05 < 0.10
    SPACE < 0.2 WIDTH1 < 0.05
}
```

The first and fourth measurement rules measure spacings on the intervals shown, between polygons that meet the WIDTH1 criteria. The second and third rules measure spacings on the intervals shown between polygons that meet the WIDTH1 and WIDTH2 criteria, respectively.

### Example 2

Consider this design rule:

Metal 1 spacing and width must be at least 0.05 in all cases. Spacing must be at least 0.08 between polygons of width greater than or equal to 0.10 and any other metal1 polygons. Spacing must be at least 0.06 between polygons of width greater than or equal to 0.05, but less than 0.10, and any other metal1 polygons.

This rule can be checked as follows:

```
m1_table {
    TDDRC metal1
    SPACE < .05
    WIDTH < .05
    SPACE < .08 WIDTH2 >= 0.10
    SPACE < .06 WIDTH2 >= 0.05 < .10
}
```

# Text

Specification statement

**TEXT** [*name*] *x* *y* [*layer*]

## Parameters

- *name*  
An optional name of a text label. If you include this parameter, you must also include the *layer* parameter. If you omit this parameter, you must also omit the *layer* parameter.
- *x* *y*  
A required pair of floating-point numbers in user units, which specify a top-level (x, y) coordinate pair.
- *layer*  
An optional original layer or layer set.

## Description

Allows free-standing, connectivity extraction text to be specified directly in the rule file as discussed in the section “Use of Text in Calibre Applications” in the *Calibre Verification User’s Manual*. This statement also allows text objects read from the layout database to be edited internally.

A Text specification statement with a specified label *name* results in a label object being generated by the connectivity extractor. The label has the specified *name*, *x* *y* location, and is assigned to *layer* when all of these elements are specified. The *layer* parameter does not need to be a [Text Layer](#) parameter, nor does it need to appear in an [Attach](#) statement.

If *layer* is a simple layer name, then the semantics are equivalent to the use of its layer number in the statement. If *layer* is a layer set name, then the semantics are equivalent to the statement being repeated for each element of the layer set.

Omitting the net *name* (and *layer*) serves to cancel all database text objects at the specified *x* *y* location. Including *name* and *layer* serves to rename a coincident text object at the specified location and on the specified layer.

Text objects from this statement are generated at the specified coordinates in user units regardless of changes to the rule file [Precision](#). Text objects generated by this statement are affected by [Layout Magnify](#) with an explicit *value*; the **AUTO** keyword has no effect, however.

There are no restrictions on the number of Text specification statements. The text objects defined in the rule file using this statement support connectivity extraction and cannot be used with the (Not) [With Text](#) or [Expand Text](#) operations.

Differences between Text and [Layout Text](#) specification statements are discussed under Layout Text.

## Text

---

In Pyxis Layout, text from the rule file is used for naming nets, net properties, and database free-floating text. Text in the rule file overrides any database free-floating text found at the same location.

See also [Layout Text File](#), [Virtual Connect Colon](#) and [Virtual Connect Name](#).

## Examples

```
TEXT "abc" 25 35 20 // add text "abc" at x=25 y=35 layer 20
TEXT 20 30           // override database text at x=20 y=30
```

## Text Depth

Specification statement

**TEXT DEPTH {PRIMARY | ALL | *number*}**

### Parameters

- **PRIMARY**

Keyword that specifies only connectivity extraction text objects from the top-level cell are selected. This is the default behavior if you do not include this statement in the rule file.

- **ALL**

Keyword that specifies that free-standing text objects from throughout the hierarchy are used as top-level connectivity extraction text.

- ***number***

Non-negative integer, variable, or numeric expression, which instructs the tool to use free-standing text objects from ***number*** levels below the top-level cell. Specifying zero is equivalent to **PRIMARY**.

### Description

Specifies hierarchical depth for selecting connectivity extraction text objects from the layout database to be used as top-level text. This statement supports connectivity extraction only. It does not influence text objects used by the (Not) [With Text](#) operation.

When specifying **ALL** or ***number***, text objects that come from lower levels of the hierarchy than the top level are transformed to the top-level coordinate space and are replicated according to the hierarchical structure of the design. Such text objects then behave as if they originated at the top level; this is true in flat as well as hierarchical applications.

In flat applications and in the nmDRC-H application, only those database text objects that are selected by this statement are used in the connectivity extractor.

In nmLVS-H, text objects from all levels of the design hierarchy are used as local text in the cells in which they appear, regardless of the Text Depth specification statement. Text objects selected by this statement serve as top-level text in addition to any local role they may perform.

This statement controls database text objects only, namely: layout database text objects and text objects entered with [Layout Text](#) or [Layout Text File](#) specification statements (which behave just as if they were database text). It does not apply to text objects entered with [Text](#) specification statements.

Pyxis Layout free-standing text objects are affected by this statement, but Pyxis Layout net properties (which are read only from the top level cell) are not. Note that Direct mode ICtrace connectivity extraction always reads text objects only from the top-level cell, regardless of Text Depth specification statements.

See also [DRC Cell Text](#).

### Examples

```
// set the text depth using the level environment variable.  
VARIABLE level ENVIRONMENT  
TEXT DEPTH level
```

# Text Layer

Specification statement

**TEXT LAYER** *layer* [*layer* ...]

## Parameters

- *layer*

A required layer name or number in the database from which to read connectivity extraction text. You can specify *layer* any number of times in one statement. Each *layer* specifies the name or number of an original layer. Layer sets are allowed and are equivalent to specifying each member of the set separately.

## Description

Specifies the layers in the layout database from which text is read for connectivity extraction. This statement may be specified any number of times. See “Use of Text in Calibre Applications” in the [Calibre Verification User’s Manual](#) for more information.

The connectivity extractor uses only those text objects having layers that appear in Text Layer specification statements. Thus, if there are no Text Layer specification statements in the rule file, then no layout database text objects are used by the connectivity extractor, and there will be no net names assigned from such objects.

This statement controls database text objects only, namely: geometric text objects and text objects entered with [Layout Text](#) or [Layout Text File](#) specification statements (which behave just as if they were database text). It does not apply to text objects entered with [Text](#) specification statements.

This statement applies to Pyxis Layout text objects, but it does not apply to Pyxis Layout net properties.

This statement supports connectivity extraction only; it does not influence text objects used by [With Text](#) or [Expand Text](#) operations in the rule file. To name ports, use the [Port Layer Text](#) statement.

Calibre expands text basepoints by 2 dbu in all directions. This affects the extent of text layers.

See also [Attach](#), [Label Order](#), [Text Depth](#), and [DRC Cell Text](#).

## Examples

```
// use these layers to specify text objects
TEXT LAYER metal1 metal2 metal3 50
```

## Text Print Maximum

Specification statement

**TEXT PRINT MAXIMUM {ALL | *number*}**

Used only by Calibre.

### Parameters

- **ALL**

Keyword that specifies that all text objects and ports are printed to the transcript. This is the default if you do not specify this statement in your rule file.

- ***number***

Positive integer that specifies the specific number of text objects and ports printed to the transcript.

### Description

Controls the maximum number of text objects and ports printed in the transcript of a Calibre run.

By default, all connectivity extraction text objects to be used in the run as well as [With Text](#) objects and ports are printed to the transcript. The output appears near the end of the layout data input module. Because the number of text objects can be excessive in certain cases, a positive integer can be used to limit the number of text objects printed in each block.

A warning is issued whenever the number of objects printed is less than the total number which could have been printed. This statement may be specified once in your rule file.

### Examples

```
TEXT PRINT MAXIMUM 100
```

## Title

Specification statement

**TITLE** *name*

### Parameters

- *name*

A required parameter specifying the title of the rule file.

### Description

Defines the title of a rule file. This statement can appear only once.

You can attach the title of a rule file to the output in the [DRC Results Database](#). The title of a rule file also appears in generated summary reports.

### Examples

```
TITLE "Design rule file for process CMOS_6m1p_diode"
```

# Topex

Layer operation

## TOPEX *layer*

Used only by Pyxis Layout.

### Parameters

- *layer*

A required original layer or layer set.

### Description

Generates a derived polygon layer containing all of the shapes on *layer* that have an external aspect in the top-level cell of the active window.

### Examples

```
// Check that no sub-cell perimeter is outside of the top-
// level cell. Cell perimeters are stored on a special layer
// called "perimeter" as objects having both internal and
// external aspect.
perm_check { X = TOPEX perimeter
// The perimeter of the top-level template.
perimeter NOT X
}
```

# Touch

Layer operation

**TOUCH** *layer1 layer2* [*constraint* [BY NET] [EVEN | ODD]]

## Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon layer.
- ***layer2***  
A required original layer or layer set, or a derived polygon layer.
- ***constraint***  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. A constraint specifies the number of *layer2* polygons a *layer1* polygon must touch in order to be selected.
- **BY NET**  
An optional keyword, used with the *constraint* parameter, which specifies that a *layer1* polygon is selected when a number of distinct nets in the set of *layer2* polygons, which have a coincident outside edge with the *layer1* polygon, meets the specified *constraint*. The connectivity of *layer2* is required and is checked at compilation time.
- **EVEN**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an even number. May not be used with ODD.
- **ODD**  
An optional keyword used with the *constraint* parameter that modifies the selection criterion of the *constraint*. A *layer1* polygon is selected if the number of *layer2* polygons that meet the *constraint* is also an odd number. May not be used with EVEN.

## Description

Selects all *layer1* polygons that lie completely outside all *layer2* polygons, but share a coincident edge, or segment, with a polygon from *layer2*. Can also select all *layer1* polygons that touch a specified number of *layer2* polygons. If either input layer is empty, there is no output.

## BY NET Keyword

If BY NET is specified, the *constraint* applies to the number of *layer2* polygons on distinct nets, in which case *layer1* polygons that meet the *constraint* are output.

## EVEN and ODD Keywords

The EVEN and ODD keywords modify the interpretation of the *constraint*. For example:

**layer1 TOUCH layer2 >3 <9 EVEN**

This operation selects polygons from layer1 that touch four, six, or eight polygons from layer2.

A constraint must be specified with these keywords. It is not useful to specify these keywords if your *constraint* does not allow for the number of polygons to meet the even or odd criterion, such as with == constraints.

See also [Not Touch](#) and [Touch Edge](#).

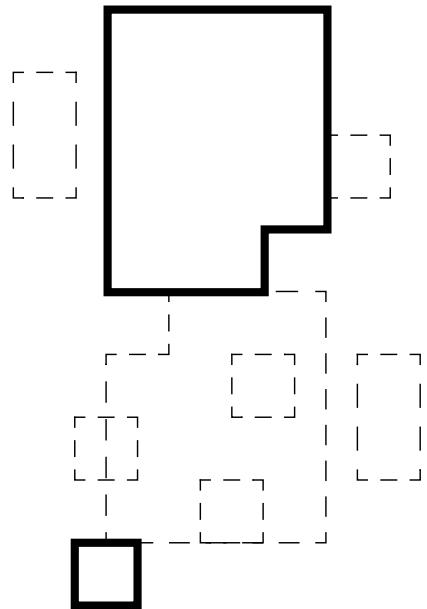
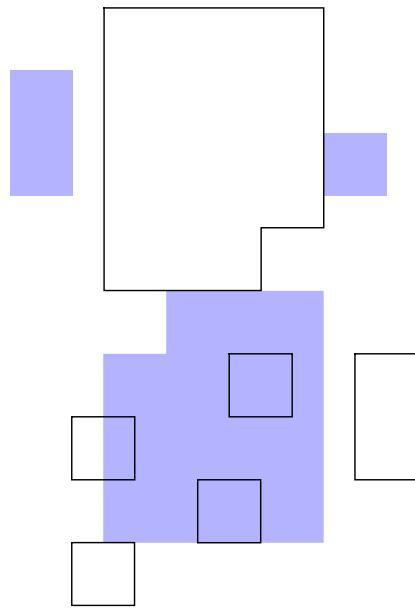
## Examples

Figure 4-327 shows two Touch operations. The operation on the left selects all layer1 polygons that lie completely outside all layer2 polygons but touch at least one layer2 polygon. The operation on the right selects all layer1 polygons that lie completely outside all layer2 polygons, but touch more than one layer2 polygon.

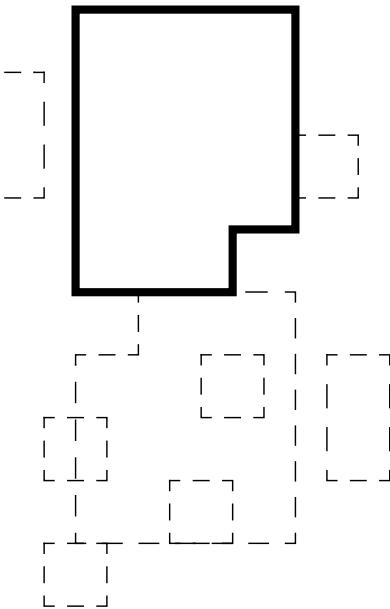
**Figure 4-327. Touch**

layer1

layer2



touch layer1 layer2  
//also layer1 touch layer2



touch layer1 layer2 > 1

# Touch Edge

Layer operation

**TOUCH EDGE *layer1 layer2* [ENDPOINT {ALSO | ONLY}]**

## Parameters

- ***layer1***  
A required original layer or layer set, or a derived polygon or edge layer.
- ***layer2***  
A required original layer or layer set, or a derived polygon or edge layer.
- **ENDPOINT {ALSO | ONLY}**  
Optional keyword group that specifies edges are selected from *layer1* that are collinear with and touch *layer2* edges only at their end points. ENDPOINT ALSO specifies to select such *layer1* edges in addition to the default output from this operation. ENDPOINT ONLY specifies to select only *layer1* edges that have this end point behavior; no other edges are output.

## Description

Selects all complete *layer1* edges that are coincident with any portion of *layer2* edges. Intersection at end points does not constitute touching by default. This is similar to the [Coincident Edge](#) operation; however, Touch Edge selects only complete edges.

The ENDPOINT ALSO and ENDPOINT ONLY options change the default behavior and allow selection of *layer1* edges that touch *layer2* edges only at their end points.

See also [Touch Inside Edge](#), [Touch Outside Edge](#), and [Not Touch Edge](#).

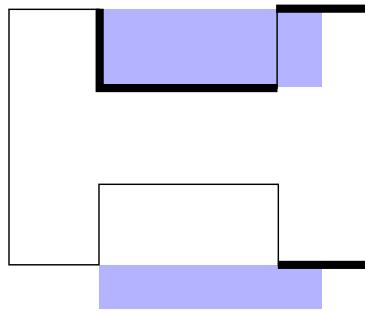
## Examples

### Example 1

The Touch Edge operation shown in Figure 4-328 selects all layer1 edges that touch layer2 edges.

**Figure 4-328. Touch Edge**

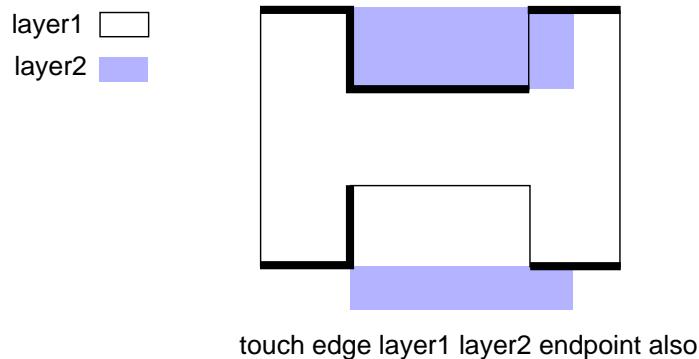
layer1   
layer2 



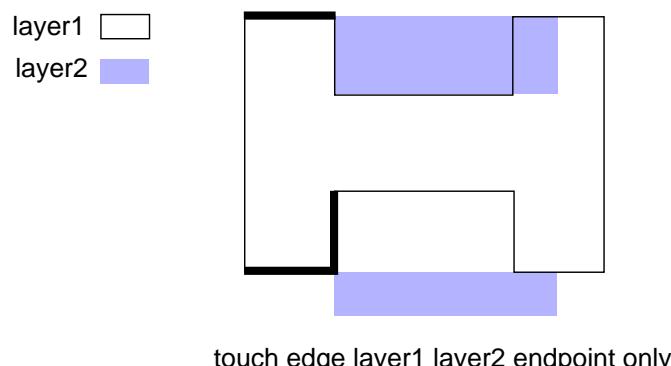
touch edge layer1 layer2  
//also layer1 touch edge layer2

**Example 2**

The Touch Edge ENDPOINT ALSO operation shown in Figure 4-329 selects all layer1 edges that touch layer2 edges, in addition to selecting layer1 edges that touch layer2 edges at their end points.

**Figure 4-329. Touch Edge ENDPOINT ALSO****Example 3**

The Touch Edge ENDPOINT ONLY operation shown in Figure 4-330 selects only the layer1 edges that touch layer2 edges at their end points.

**Figure 4-330. Touch Edge ENDPOINT ONLY**

## Touch Inside Edge

Layer operation

**TOUCH INside EDGE *layer1* *layer2*** [ENDPOINT {ALSO | ONLY}]

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.
- ENDPOINT {ALSO | ONLY}

Optional keyword group that specifies edges are selected from *layer1* that are collinear with and touch *layer2* edges only at their end points. ENDPOINT ALSO specifies to select such *layer1* edges in addition to the default output from this operation. ENDPOINT ONLY specifies to select only *layer1* edges that have this end point behavior; no other edges are output.

### Description

Selects all complete *layer1* edges coincident with *layer2* edges, where the polygons share internal area. Intersection at end points does not constitute touching by default. This is similar to the [Coincident Inside Edge](#) operation; however, Touch Inside Edge selects only complete edges.

The ENDPOINT ALSO and ENDPOINT ONLY options change the default behavior and allow selection of *layer1* edges that touch *layer2* edges only at their end points.

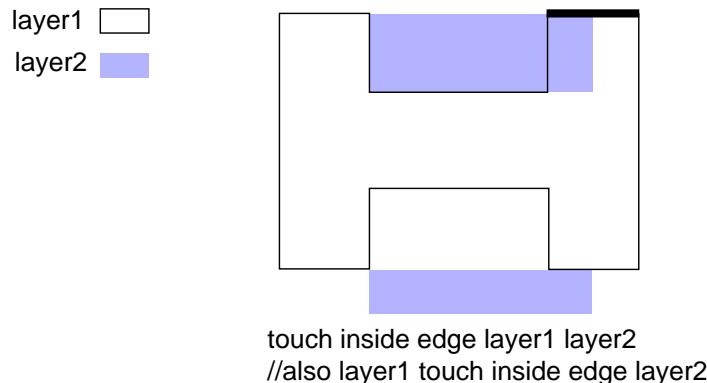
See also [Touch Edge](#), [Touch Outside Edge](#), and [Not Touch Inside Edge](#).

### Examples

#### Example 1

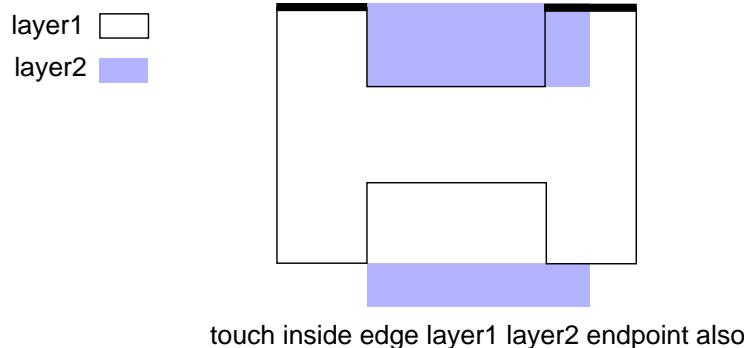
The Touch Inside Edge operation shown in Figure 4-331 selects the *layer1* edge segments that touch *layer2* edges, where the polygons share internal area.

**Figure 4-331. Touch Inside Edge**

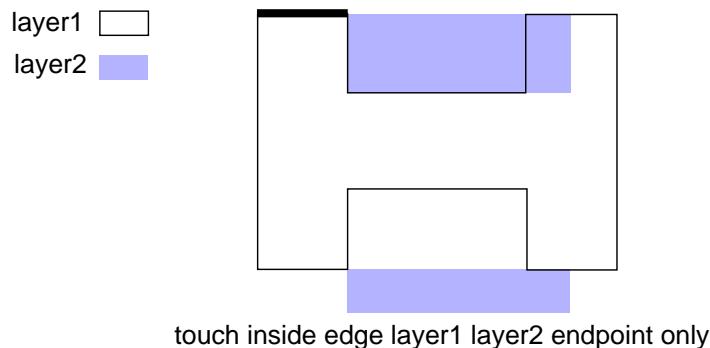


**Example 2**

The Touch Inside Edge ENDPOINT ALSO operation shown in Figure 4-332 selects the layer1 edge segments that touch layer2 edges where the polygons share internal area. Layer1 edges that touch layer2 edges at their end points are also selected.

**Figure 4-332. Touch Inside Edge ENDPOINT ALSO****Example 3**

The Touch Inside Edge ENDPOINT ONLY operation shown in Figure 4-333 selects only the layer1 edge segment that are collinear with and touch layer2 edges at their end points.

**Figure 4-333. Touch Inside Edge ENDPOINT ONLY**

# Touch Outside Edge

Layer operation

**TOUCH OUTside EDGE *layer1 layer2* [ENDPOINT {ALSO | ONLY}]**

## Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon or edge layer.
- *layer2*  
A required original layer or layer set, or a derived polygon or edge layer.
- ENDPOINT {ALSO | ONLY}

Optional keyword group that specifies edges are selected from *layer1* that are collinear with and touch *layer2* edges only at their end points. ENDPOINT ALSO specifies to select such *layer1* edges in addition to the default output from this operation. ENDPOINT ONLY specifies to select only *layer1* edges that have this end point behavior; no other edges are output.

## Description

Selects all complete *layer1* edges that are coincident with *layer2* edges, and where the polygons do not share internal area. Intersection at end points does not constitute touching by default. This is similar to the [Coincident Outside Edge](#) operation; however Touch Outside Edge selects only complete edges.

The ENDPOINT ALSO and ENDPOINT ONLY options change the default behavior and allow selection of *layer1* edges that touch *layer2* edges only at their end points.

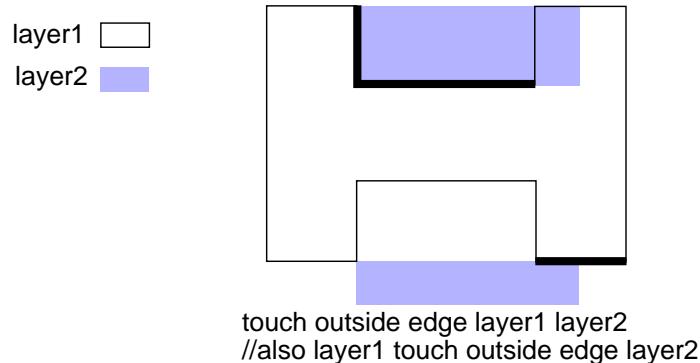
See also [Touch Edge](#), [Touch Inside Edge](#), and [Not Touch Outside Edge](#).

## Examples

### Example 1

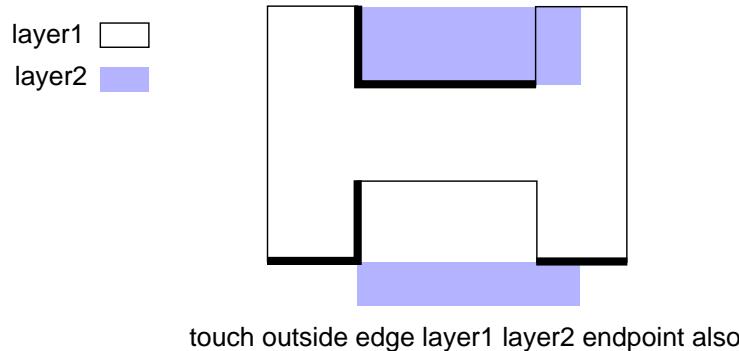
The Touch Outside Edge operation shown in Figure 4-334 selects the layer1 edge segments that touch outside layer2 edges.

**Figure 4-334. Touch Outside Edge**



**Example 2**

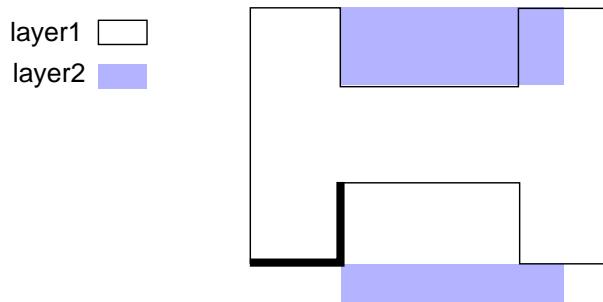
The Touch Outside Edge ENDPOINT ALSO operation shown in Figure 4-335 selects the layer1 edge segments that touch outside layer2 edges, in addition to selecting layer1 edges that touch layer 2 edges at their end points.

**Figure 4-335. Touch Outside Edge ENDPOINT ALSO**

touch outside edge layer1 layer2 endpoint also

**Example 3**

The Touch Outside Edge ENDPOINT ONLY operation shown in Figure 4-336 selects only the layer1 edge segments that touch outside layer2 edges at their end points.

**Figure 4-336. Touch Outside Edge ENDPOINT ONLY**

touch outside edge layer1 layer2 endpoint only

# Trace Property

Specification statement

```
TRACE PROPERTY {component_type [‘(‘component_subtype‘)’]}  
  {source_property [‘(‘source_spice_parameter‘)’]}  
  {layout_property [‘(‘layout_spice_parameter‘)’]}  
  [[[tolerance1 [‘(‘tolerance1_property‘)’] [tolerance2 [‘(‘tolerance2_property‘)’]]]  
  [ABSOLUTE]}  
  | NO  
  | STRING [NO]  
  [CELL LIST list_name]  
  [DIRECT] [MASK]} |  
  [‘[‘ trace_property_computation ‘]’]
```

## Summary

Enables tracing and comparison of **Device** properties between source and layout netlists during LVS. In ICverify, this statement corresponds to Pyxis Layout functions \$trace\_property\_numeric() or \$trace\_property\_string(), depending on whether or not the STRING keyword is specified.

## Parameters

- ***component\_type***

A required SPICE component type (device element name) to which this statement applies.

In ICverify, it corresponds to *component\_type* in \$trace\_property\_numeric( ) or \$trace\_property\_string( ) functions.

- **(*component\_subtype*)**

An optional string, which must be in parentheses, that specifies the SPICE component subtype (or model name) to which this statement applies. The default *component\_subtype* is a null string.

In ICverify, a *component\_subtype* corresponds to a *comp\_subtype* that appears in a \$trace\_property\_numeric() or \$trace\_property\_string() function for this component type.

LVS expects the component subtypes (or model names) in your source netlist to conform to the naming conventions listed in the layout netlist. When using nmLVS for layout netlist extraction, these subtypes are as specified in your **DEvice** statements. If your source netlist does not conform to these conventions, your device properties may not be traced.

- ***source\_property***

A required source property name to be traced in LVS. It may be numeric or string. If string, then the STRING keyword must be specified.

In ICverify, it corresponds to *source\_prop\_name* in a \$trace\_property\_numeric() or \$trace\_property\_string() function.

- ***layout\_property***

A required layout property name to be traced in LVS. It may be numeric or string. If string, then the STRING keyword must be specified.

The values of the specified property in layout device instances are compared to the values of the ***source\_property*** name on corresponding source instances.

In ICverify, this corresponds to the *mask\_prop\_name* and *direct\_prop\_name* arguments.

- **(*source\_spice\_parameter*) and (*layout\_spice\_parameter*)**

An optional upper- or lowercase letter, which must be in parentheses. The letter indicates the name of a SPICE parameter to be extracted from values of the ***source\_property*** or ***layout\_property***, respectively (see section “SPICE-Like Property Syntax” in the *ICverify Manual*). Possible strings are as follows:

<b>W</b> MOS width	<b>C</b> capacitance
<b>L</b> MOS length	<b>A</b> diode area
<b>M</b> multiplier factor	<b>P</b> diode perimeter
<b>R</b> resistance	<b>parnam</b> arbitrary parameter name set to a value

These parameters may not be specified if STRING is specified.

In ICverify, these parameters corresponds to the optional SPICE parameter name suffix of the *source\_prop\_name* and *layout\_prop\_name* arguments in a \$trace\_property\_numeric() or \$trace\_property\_string() function.

- ***tolerance1* [(*tolerance1\_property*)]**

An optional set of parameters, as follows:

*tolerance1* — A positive, floating-point number that specifies the tolerance in percent for reporting property value discrepancies. Discrepancies are reported when they exceed the specified tolerance, either above or below the source property value. If specified with *tolerance2*, then *tolerance1* is the tolerance allowed below the source property value and *tolerance2* is the tolerance allowed above the source property value. The default behavior is *tolerance1* equal to zero if you do not include either *tolerance1* or STRING in the rule file. The tolerance parameters may be expressed as variables.

In ICverify, *tolerance1* corresponds to the *tolerance* parameter for the \$trace\_property\_numeric function.

(*tolerance1\_property*) — An optional source property parameter that is used for instance-specific tolerance values. It is only used if *tolerance1* is specified, and it must be enclosed in parentheses. Such properties are expected to have floating-point values in percent.

If the *tolerance1\_property* parameter is indicated and this property is found on a source instance, then the value of this property is used in place of the original *tolerance1* value for that instance. If *tolerance1\_property* is indicated but is not found on a source

instance, or does not have a valid numeric value, then the original *tolerance1* value is used instead.

- *tolerance2* [(*tolerance2\_property*)]

An optional set of keywords, as follows:

*tolerance2* — A positive, floating-point number in percent that specifies the tolerance for reporting discrepancies above the source property value. This parameter may only be specified if *tolerance1* is specified. The tolerance parameters may be expressed as variables. If not specified, *tolerance2* defaults to *tolerance1*.

(*tolerance2\_property*) — An optional source property parameter that is used for instance-specific tolerance values. It is only used if *tolerance2* is specified and it must be enclosed in parentheses. Such properties are expected to have floating-point values in percent.

If the *tolerance2\_property* parameter is indicated and this property is found on a source instance, then the value of this property is used in place of the original *tolerance2* value for that instance. If *tolerance2\_property* is indicated but is not found on a source instance, or does not have a valid numeric value, then the original *tolerance2* value is used instead.

- ABSOLUTE

An optional keyword specified with the *tolerance1* and *tolerance2* values, and causes them to be treated as absolute quantities rather than as relative percentages. This applies to all numeric tolerance values, as well as to any instance-specific tolerance values in the input database, as specified by the *tolerance1\_property* and *tolerance2\_property* parameters.

- NO

An optional keyword that must be specified with a CELL LIST parameter. It causes property tracing not to occur in the cells specified in the list. The NO keyword cannot be specified with the *tolerance* parameters.

- STRING

An optional keyword that treats the values of *source\_property* and *layout\_property* as character strings. The properties are compared as strings. This keyword cannot be specified with the *tolerance* parameters.

- CELL LIST *list\_name*

An optional keyword and parameter that specifies a cell list from a [LVS Cell List](#) specification statement. This is used for cell overrides of Trace Property rules and is used only by Calibre.

- DIRECT

An optional keyword that places the operation in the Direct verification set. This keyword applies only to ICtrace. This option is used by default.

- **MASK**

An optional keyword that places the operation in the Mask verification set. This keyword applies only to Calibre. This option is used by default.

The use of the DIRECT versus MASK set in a Trace Property statement is as follows:

- In Calibre, the Mask set of Trace Property specification statements are used with *source\_property* applying to the [Source System](#) and *layout\_property* applying to the [Layout System](#). Note that layout devices may be extracted in this scenario.
- In ICtrace Direct mode LVS, the Direct set of Trace Property specification statements are used with *source\_property* applying to the schematic and *layout\_property* applying to the layout. Note that layout devices are not extracted in this scenario.
- In ICtrace Mask mode LVS, the Mask set of Trace Property specification statements are used with *source\_property* applying to the schematic and *layout\_property* applying to the layout. Note that layout devices are extracted in this scenario.

- **[ ‘*trace\_property\_computation* ’ ]**

A procedure that defines the operations to take place when values of properties on layout instances are compared to values of corresponding properties on source instances. The procedure must appear between square brackets [ ]. If specified, no other parameters except the *component\_type* and *component\_subtype* may be specified. Only one property computation procedure may be specified per Trace Property statement.

The procedure lists the names and types of the properties to be made available, and specifies the method of computation and discrepancy reporting to be used with available instance data. This procedure uses a built-in language described under “[Trace Property Computation and Reporting Language](#)” on page 1872.

## Description

This statement controls how device properties are compared between layout and source netlists. During LVS comparison, the *source\_property* is compared to the *layout\_property* for a given *component\_type* and optional *component\_subtype*. Discrepancies are shown in the LVS Report. Note that property differences between the source (base) value and the layout value are not reported if the difference is less than 1E-05 \* base value. This is done because it prevents false errors that could be caused by binary numeric representation differences in host hardware.

Property names must begin with a letter character followed any legal SPICE characters. See “[General SPICE Syntax Summary](#)” in the *Calibre Verification User’s Manual* for more details.

The *source\_property* names are taken from the source design.

When extracting the layout netlist using Calibre nmLVS, the layout property names you specify in a Trace Property statement should come from the property computations in a Device statement (which may be the defaults), *not the property names that appear in your SPICE netlist*. See [Table 4-5](#) on page 196 for a listing of the default layout property names available for tracing. Default properties available for tracing are discussed under “[Default Property](#)

Computations” in the [Calibre Verification User’s Manual](#). Default Trace Property names are also discussed in the tables under “Element Statements” in the same manual.

A null *component\_subtype* indicates a fall-through rule. Such a rule applies to instances of the specified *component\_type* that have no subtype, or that have a subtype that does not explicitly appear in any Trace Property rule that applies to the *component\_type*. For example, suppose you have these Trace Property statements:

```
TRACE PROPERTY R(nwell) r r 0
TRACE PROPERTY R(poly)   r r 0
TRACE PROPERTY R          r r 0 // fall-through rule
```

The final Trace Property statement applies to all R devices that do not have the nwell or poly subtypes.

For a given device type and subtype, Trace Property statements that match both *component\_type* and *component\_subtype* are applied, such as the first two Trace Property statements above. If no statement that matches both *component\_type* and *component\_subtype* is found, then statements that match a *component\_type* but have no *component\_subtype* are applied, such as the final statement above. If a device has no subtype, only statements that have no *component\_subtype* are applied, such as the final statement above.

This behavior applies across all *traced* properties for a given *component\_type*. For example, if you have a subtype-specific rule for property x of R(nwell) but not for property y, then there is no fall-through check for y that corresponds to R with no subtype.

Given a layout device and a source device that are matched to each other, the following criteria are used to decide which Trace Property statements to apply to this device pair:

- If layout and source devices have the same subtype, empty or not, that subtype is used.
- If one of the devices has an empty subtype and the other one has a non-empty subtype, and [LVS Strict Subtypes YES](#) or [LVS Exact Subtypes YES](#) are not specified, then the non-empty subtype is used.

If the device with the empty subtype does not have the same properties as the device with a non-empty subtype, then missing property errors are issued for properties that are traced for the non-empty subtype but not for the empty subtype. These errors are reported at the top of the LVS Report, whether or not any devices with non-empty subtypes were actually matched to any devices with empty subtypes. For example, assume you have this in both your layout and source netlists:

```
R0 1 2 nwell r=1 t=1
R1 2 3 r=2 t=2
R2 3 4 r=3
```

and you have this in your rule file:

```
TRACE PROPERTY R(nwell) r r 0
TRACE PROPERTY R(nwell) t t 0
TRACE PROPERTY R r r 0
```

You would get these errors:

```
Error: Properties missing on instances in layout.
Error: Properties missing on instances in source.
```

and the LVS Report would show this for both layout and source:

```
Properties Missing on Instances:
  1   property t           not found on      R2 (R)
```

- If the two matched devices have different non-empty subtypes, then a “bad component subtype” error is reported and properties are not compared at all. The same applies to any different subtypes, empty or not, if [LVS Strict Subtypes YES](#) or [LVS Exact Subtypes YES](#) are specified.

Recall that the standard MP and MN devices defined with SPICE M statement, such as this:

```
M1 d g s b p l=1 w=2
```

have non-empty subtypes P and N, respectively. To define an MP device with an empty subtype one has to use the X statement:

```
.subckt mp d g s b
.ends
x1 d g s b mp l=1 w=2
```

A given set of *component\_type*, *component\_subtype*, and *source\_property* parameters may not be specified more than once within any Trace Property statement within each of the four types of Trace Property statements (numeric Mask mode, string Mask mode, numeric Direct mode, and string Direct mode). The optional source *spice\_value* is treated as part of the *source\_property* to which it applies in the comparison. If *component\_subtype* is unspecified, only *component\_type* and *source\_property* are used in the comparison.

Comparison of *component\_type* and *component\_subtype* are case-sensitive if you specify [LVS Compare Case YES](#), [TYPES](#), or [SUBTYPES](#).

## Considerations for User Defined Trace Property Programs

The *trace\_property\_computation* allows user-defined programs for property tracing. Details of these procedures are discussed under [“Trace Property Computation and Reporting Language”](#) on page 1872. When you provide a *trace\_property\_computation* procedure, you must declare and handle all properties of interest in the procedure you provide.

Certain mixtures of Trace Property user-defined programs and built-in Trace Property statements in a rule file are disallowed when they share the same device element name. This is because property discrepancies could be incorrectly reported or missed during Calibre nmLVS comparison. Consider this example:

```
TRACE PROPERTY mp [
  property p1
  rep_num_disc(p1,999)
]

TRACE PROPERTY mp(p) p2 p2
```

The user-defined program referencing element mp cannot be used at the same time that a built-in statement is used with device mp(p). A TRP24 compiler error is issued.

The following example produces a TRP25 compiler error:

```
TRACE PROPERTY mp(p) [
property p1
rep_num_disc(p1,999)
]

TRACE PROPERTY mp p2 p2
```

## Calibre Operation

In Calibre, *source\_property* applies to the design specified with the [Source Path](#) specification statement and *layout\_property* applies to the design specified with the [Layout Path](#) specification statement. Note that the term layout is used loosely here; source and layout can actually be designs in any form, such as SPICE, GDSII, OASIS, or Cnet, as specified with the [Source System](#) and [Layout System](#) specification statements, respectively.

## ICVerify Operation

In ICVerify, if the rule file contains at least one Trace Property specification statement, then all existing trace property rules are removed from the Pyxis Layout session memory and the new set of rules specified in the rule file is stored in the session. If the Mask keyword is specified, then *layout\_property* sets the *mask\_prop\_name* field in the respective rule in Pyxis Layout. If the Direct keyword is specified, then *layout\_property* sets the *direct\_prop\_name* field in the respective rule in Pyxis Layout. If both Mask and Direct are specified, or if both are omitted, then *layout\_property* sets both the *direct\_prop\_name* and *mask\_prop\_name* fields in Pyxis Layout. If the rule file contains no Trace Property specification statements, then trace property rules currently stored in the Pyxis Layout session are not affected.

In ICVerify, if a given set of *component\_type*, *component\_subtype*, and *source\_property* exists in both a numeric Mask and numeric Direct mode Trace Property statement, then the *tolerance* parameters must agree. The optional source *spice\_value* is treated as part of the *source\_property* to which it applies in the comparison. If *component\_subtype* is unspecified, only *component\_type* and *source\_property* are used in the comparison.

## Built-in Property Classification

For device reduction and other processing, the LVS circuit comparison module recognizes certain property names as built-in properties. For example, LVS computes effective values for built-in properties when it reduces devices in series and parallel.

Default properties are classified based on their names as they are used in the corresponding Device statement, specifically: W, L, AS, AD, PS, PD, C, R, A, P, and M, which denote width, length, area of source, area of drain, perimeter of source, perimeter of drain, capacitance, resistance, area, perimeter, and multiplier factor, respectively. This convention is used in the layout as well as in the source.

This example shows the difference between how diode properties are represented in a SPICE netlist and how they are represented in Trace Property statements. Typically diodes are represented in SPICE format as:

```
.subckt PWELD out1 in1 a=1 p=1
...
D0 out1 in1 PWELD AREA=a PJ=p
.end
```

The correct way to trace these properties is as follows:

```
TRACE PROPERTY D (PWELD) A A
TRACE PROPERTY D (PWELD) P P
```

as opposed to:

```
//TRACE PROPERTY D (PWELD) "AREA" "AREA"
//TRACE PROPERTY D (PWELD) PJ PJ
```

In ICtrace, Trace Property statements (or the \$trace\_property\_numeric() setup function) can override this built-in naming convention and can specify different property names in the source or in Direct mode layout. For example, the statement:

```
TRACE PROPERTY MP(X) WIDTH W 0
```

implies that for elements of type MP(X), the width property in the source is WIDTH (to have this effect in ICtrace, the Trace Property statement must apply to the Mask mode). The pair of statements:

```
TRACE PROPERTY MP(X) WIDTH W 0 MASK
TRACE PROPERTY MP(X) WIDTH ICW 0 DIRECT
```

implies that for elements of type MP(X), the width property in the source is WIDTH and the width property in Direct mode layout is ICW. The Pyxis Layout function call:

```
$trace_property_numeric("MP", "X", "WIDTH", "ICW", "W", 0, @true);
```

also implies, as before, that for elements of type MP(X), the width property in the source is WIDTH and the width property in Direct mode layout is ICW.

## Tracing Properties AS, AD, PS, and PD

The properties AS, AD, PS, and PD in MOS transistors represent area of source, area of drain, perimeter of source and perimeter of drain respectively. These properties require special handling because source and drain pins may be swapped. When source and drain pins are swapped (the layout source pin corresponds to the schematic drain pin and vice versa), then LVS correctly swaps the values of AS and AD, and the values of PS and PD, before comparing these properties in source and layout. This swapping is done in standalone MOS devices as well as MOS devices that form logic gates. In the latter case, the decision is based on the position of the source and drain pins relative to the output pin or pins of the gate.

## Tolerances

Either one or two tolerances for discrepancy reporting may be used. If only *tolerance1* is specified, then discrepancies that exceed the tolerance, either under or over the *source\_property* value, are reported. If *tolerance1* and *tolerance2* are used, then the default meanings are:

*tolerance1* = under%

*tolerance2* = over%

In other words, the layout value can be up to *tolerance1*% smaller than the source value or up to *tolerance2*% larger than the source value. More accurately, consider a layout property value L and source property value S. The tolerance condition is satisfied under these conditions:

when two values are specified

$-tolerance1 \leq 100 * (L - S) / S \leq tolerance2$

when one value is specified (equivalent to the old definition)

$-tolerance1 \leq 100 * (L - S) / S \leq tolerance1$

Additionally, *tolerance\_property1* and *tolerance\_property2* can be specified to provide instance-specific control of tolerances. Examples are shown later in this section.

The optional ABSOLUTE keyword specifies that tolerance parameters are to be treated as absolute quantities rather than relative percentage values. In other words, the layout value can be up to *tolerance1* smaller than the source value or up to *tolerance2* larger than the source value (as opposed to tolerance1% smaller and tolerance2% larger normally). Recall that *tolerance2* defaults to *tolerance1* when only one tolerance parameter is specified.

More precisely, consider a layout property value L and source property value S, and assume that the ABSOLUTE keyword is specified. When both *tolerance1* and *tolerance2* parameters are specified, the tolerance condition is satisfied in this case:

$-tolerance1 \leq L - S \leq tolerance2$

When only one tolerance parameter is specified, the tolerance condition is satisfied in this case:

$-tolerance1 \leq L - S \leq tolerance1$

When the ABSOLUTE keyword is specified, the units and scale of the tolerance parameters must match the units and scale of numbers in the input database. For example, if the source database contains values such as W=5E-6 or W=5U, then an absolute tolerance of 0.1E-6 for W would make sense, but an absolute tolerance of 0.1 would not (because it is too large by factor of a million). Conversely, if the source database contains values such as W=5, then an absolute tolerance of 0.1 would make sense. To summarize:

```
TRACE PROPERTY MP W W 0.1E-6 ABSOLUTE // OK for W=5E-6 or W=5U
TRACE PROPERTY MP W W 0.1 ABSOLUTE // OK for W=5
```

To protect against missed property errors, LVS performs a sanity check of absolute tolerances as follows:

- For any Trace Property statement with ABSOLUTE tolerances, if more than 50% of the values found for that property in the source are smaller than one tenth of the tolerance, then the tolerance value is assumed to be wrong.
- When two, distinct tolerance values are specified, the check is performed for each tolerance separately; that is, if 50% or more of the values found for the property in the source are smaller than one tenth of *tolerance1*, or if 50% or more of the values found for the property in the source are smaller than one tenth of *tolerance2*, then one or both tolerances are assumed to be wrong.

When this happens, the overall LVS result is set to INCORRECT, and the following secondary comparison status is reported:

Error:      Property tolerances out of range of actual values.

In addition, the incorrect Trace Property statements are listed in a special section of the LVS report titled ABSOLUTE TRACE PROPERTY TOLERANCES OUT OF RANGE.

You can control whether absolute tolerance checks are made for properties that equal 0 with the [LVS Out Of Range Exclude Zero](#) specification statement.

## Cell-Specific Trace Property Rules

The optional CELL LIST parameter specifies the name of a cell list from a LVS Cell List specification statement. It indicates that this particular Trace Property rule applies only to cells in the list and overrides other similar Trace Property rules in those cells.

More specifically, when the rule file is compiled, all Trace Property rules without CELL LIST parameters are collected first. These are the *basic rules*. Next, the following is done for each Trace Property rule that has a CELL LIST parameter (the cell-specific rules):

- For the indicated cells, the cell-specific rule cancels any basic rule with the same *component\_type*, *component\_subtype*, and *source\_property* parameters.  
If, in the cell-specific rule, the *component\_subtype* parameter is not indicated, then the cell-specific rule cancels any basic rule with the same *component\_type*, unspecified *component\_subtype*, and the same *source\_property* parameter.
- For the indicated cells, the cell-specific rule is added to the remaining collection of Trace Property rules and is processed according to the usual rules of preference.

For the following examples, consider the statement:

```
LVS CELL LIST A c11 c12 c13
```

Assume that the only Trace Property rules present are those shown in the following cases.

For cells in list A (cells c11, c12 and c13), the rule

```
TRACE PROPERTY MN(NZ) W W 2 CELL LIST A
```

## Trace Property

---

overrides any basic rule of the form

```
TRACE PROPERTY MN(NZ) W ...
```

For cells in list A, the null-subtype rule:

```
TRACE PROPERTY MN W W 2 CELL LIST A
```

overrides any null-subtype basic rule of the form

```
TRACE PROPERTY MN W ...
```

but does not override a subtype-specific basic rule of the form

```
TRACE PROPERTY MN(NZ) W ...
```

It is best practice to use a set of basic Trace Property rules for generic cells and to use cell-specific rules only to override the basic rules, as needed.

The NO keyword is used with the CELL LIST parameter and causes property tracing not to occur in the specified list of cells.

More examples appear in the Examples section.

## Reading of W/L Partner Properties

W and L are sometimes called *partner* properties because one is often required to calculate effective values for the other during device reduction.

Calibre nmLVS automatically reads from the input database L properties if they are required for the calculation of effective W values during device reduction. L properties are read *if required*, even when they are not traced themselves. For information on this topic, see “Reading Built-In W/L Partner Properties” in the [Calibre Verification User’s Manual](#).

## Consistency Checking

This section is provided for completeness. You ordinarily will not need to be concerned with it.

For any Mask mode Trace Property specification statement, there is a set of rule file device definitions that are said to correspond to it. Given a Trace Property statement of the form:

**TRACE PROPERTY X (A) ...** (explicit component subtype A)

the set of corresponding device definitions is:

All DEVICE X ... (no model name)

All DEVICE X (A) ... (explicit model name)

Given a Trace Property statement of the form:

**TRACE PROPERTY X ...** (no component subtype)

the set of corresponding device definition is:

All DEVICE X ... (no model name)

All DEVICE X (B) ... (explicit model name)

where, in the second case, X and B are not the component type and subtype of any other Mask mode Trace Property statement.

Consistency checking is a compilation check that a property specified in a Mask mode Trace Property statement will actually be computed with correct type by some corresponding rule file Device definition. The check proceeds as follows:

- A *layout\_spice\_parameter* or *source\_spice\_parameter* cannot be specified if the property requires device recognition. This is because properties for extracted devices do not have SPICE values.
- There must be at least one **Device** operation corresponding to the Trace Property specification statement.
- The property being checked must be computed as follows:
  - In *all* Device operations that correspond to the Trace Property specification statement.
  - With the correct type (numeric or string) in *all* Device operations that correspond to the Trace Property specification statement. Currently, this means no STRING option in the Trace Property specification statement because extracted devices do not currently compute string-valued properties.
  - Not as a vector type in any Device operation that corresponds to the Trace Property statement.

The following rules govern which properties undergo consistency checking:

- In Calibre, if the Source System requires device recognition (such as GDSII, OASIS, ASCII, or binary), then consistency checking is performed on the *source\_property* of Mask mode Trace Property specification statements. Note that since GDSII and OASIS are not supported as Source System, this check is actually redundant.
- In Calibre, if the Layout System requires device recognition (such as GDSII, OASIS, ASCII, or binary), then consistency checking is performed on the *layout\_property* of all Mask mode Trace Property statements.
- In ICVerify, consistency checking is performed on the *layout\_property* of all Mask mode Trace Property specification statements.

## Examples

### Example 1

Trace property w in mp devices in layout and source with 0% error tolerance.

```
TRACE PROPERTY mp w w
```

### Example 2

Trace property w in source and layout of mp devices reporting discrepancies of more than 3%.

```
TRACE PROPERTY mp w w 3
```

## Trace Property

---

### Example 3

Allow up to 0% under, 3% over tolerances.

```
TRACE PROPERTY mp w w 0 3
```

### Example 4

Compare instpar(c) in source to c in layout of c(x) devices tolerance of 5%, use DIRECT mode ICtrace.

```
TRACE PROPERTY c(x) instpar(c) c 5 DIRECT
```

### Example 5

Do string comparison of technology and tech properties in buff devices using DIRECT mode ICtrace.

```
TRACE PROPERTY buff technology tech STRING DIRECT
```

### Example 6

W tolerance is 2%, but if the TOLW property is found on a source instance, then it determines the tolerance for that instance instead. For example, if TOLW=1 then a tolerance of 1% is used when comparing that instance to the corresponding layout instance.

```
TRACE PROPERTY MP W W 2 ( TOLW )
```

### Example 7

W tolerance-under is 2% and W tolerance-over is 3%, but if the TOLWU property is found on a source instance, then it determines the W tolerance-under for that instance. If the TOLWO property is found on a source instance, then it determines the W tolerance-over for that instance.

```
TRACE PROPERTY MP W W 2 ( TOLWU ) 3 ( TOLWO )
```

### Example 8

Rule file:

```
TRACE PROPERTY R R R 3 ( TOLRU ) 3 ( TOLRO )
```

Source:

```
.SUBCKT TOP
R1 A1 B1 R=100
R2 A2 B2 R=100 TOLRO=5
R3 A3 B3 R=100
R4 A4 B4 R=100 TOLRU=1
R5 A5 B5 R=100 TOLRU=5 TOLRO=5
.ENDS
```

Layout:

```
.SUBCKT TOP
R1 A1 B1 R=104      $ FAIL
R2 A2 B2 R=104      $ PASS
R3 A3 B3 R= 98      $ PASS
R4 A4 B4 R= 98      $ FAIL
R5 A5 B5 R=104      $ PASS
.ENDS
```

In this example, R1 fails the test because layout resistance value 104 exceeds the tolerance of 3% versus the source 100; R2 passes because the source indicates a larger tolerance-over of 5% for that instance. R3 passes because 98 is within the tolerance of 3%; R4 fails because the source indicates a tighter tolerance-under of 1%. Finally, R5 passes as well.

### Example 9

Rule file:

```
TRACE PROPERTY R      R R 3 ( TOLR )
TRACE PROPERTY R(A)  R R 3 ( TOLRA )
```

Source:

```
.SUBCKT TOP
R1 A1 B1 R=100 TOLR=5 TOLRA=2
R2 A2 B2 R=100 TOLR=5 TOLRA=2
.ENDS
```

Layout:

```
.SUBCKT TOP
R1 A1 B1 R=104      $ NO SUBTYPE - PASS
R2 A2 B2 R=104 $[A] $ SUBTYPE A - FAIL
.ENDS
```

In this example, R1 passes the test because it has no subtype in the layout or source; thus, the no-subtype TRACE PROPERTY R rule is applied, which means that the TOLR property determines tolerance. The TOLR value in the source is 5% which is larger than the difference between the layout resistance (104) and the source resistance (100). R2 fails the test because it has subtype A in the layout. According to usual Trace Property semantics, the TRACE PROPERTY R(A) rule is applied, which means that the TOLRA property determines tolerance. The TOLRA value in the source is 2%, which is smaller than the difference between the layout resistance (104) and the source resistance (100).

### CELL LIST Examples

The next examples show the use of the CELL LIST parameter for cell overrides. For the following examples, consider the statement:

```
LVS CELL LIST A c11 c12 c13
```

Assume that the only Trace Property rules present are those shown.

### Example 10

The basic tolerance for property W in MN devices is 2 percent. However, for cells in list A, a tolerance value of 3 percent is used.

```
TRACE PROPERTY MN W W 2
TRACE PROPERTY MN W W 3 CELL LIST A
```

## Trace Property

---

### Example 11

The basic tolerance for property W in MN devices is 2 percent. For cells in list A, the tolerance for MN devices with component subtype NZ is 3 percent; the tolerance for other MN devices remains 2.

```
TRACE PROPERTY MN W W 2
TRACE PROPERTY MN(NZ) W W 3 CELL LIST A
```

### Example 12

The basic tolerance for property W in MN devices is 2 percent. For MN(NZ) devices, the basic tolerance is 3 percent. For cells in list A, the tolerance for MN(NZ) devices is 5 percent; for other MN devices the tolerance remains 2 percent.

```
TRACE PROPERTY MN W W 2
TRACE PROPERTY MN(NZ) W W 3
TRACE PROPERTY MN(NZ) W W 5 CELL LIST A
```

### Example 13

The basic tolerance for property W in MN devices is 2 percent. For MN(NZ) devices, the basic tolerance is 3 percent. For cells in list A, the tolerance for MN devices other than NZ is 5 percent; for MN(NZ) devices the tolerance remains 3 percent.

```
TRACE PROPERTY MN W W 2
TRACE PROPERTY MN(NZ) W W 3
TRACE PROPERTY MN W W 5 CELL LIST A
```

### Example 14

The basic tolerance for property W in MN devices is 2 percent. For MN(NZ) devices, the basic tolerance is 3 percent. For cells in list A, the tolerance for all MN devices is 5 percent.

```
TRACE PROPERTY MN W W 2
TRACE PROPERTY MN(NZ) W W 3
TRACE PROPERTY MN W W 5 CELL LIST A
TRACE PROPERTY MN(NZ) W W 5 CELL LIST A
```

### Example 15

Property C is traced in capacitor devices in all cells. For cells in list A, both C and W are traced.

```
TRACE PROPERTY C C C 0
TRACE PROPERTY C W W 0 CELL LIST A
```

### Example 16

Property C is traced in capacitor devices in all cells. For cells in list A, property R in resistor devices is traced as well.

```
TRACE PROPERTY C C C 0
TRACE PROPERTY R R R 0 CELL LIST A
```

### Example 17

Property R is traced with a relative tolerance of 2 percent. However, for cells in list A, property R is traced with an absolute tolerance of 3.

```
TRACE PROPERTY R R R 2
TRACE PROPERTY R R R 3 ABSOLUTE CELL LIST A
```

**Example 18**

Property R is traced with a tolerance of 2 percent. However, for cells in list A, property R is traced with a tolerance of 10000. (That is, tracing of R is essentially disabled for those cells).

```
TRACE PROPERTY R R R 2
TRACE PROPERTY R R R 10000 CELL LIST A
```

**Example 19**

For cells in list A, property R in resistor devices is traced. No properties are traced in other cells.

```
TRACE PROPERTY R R R 0 CELL LIST A
```

**Example 20**

The basic tolerance for property R is 4 percent. For cells aaa, bbb, and ccc, the tolerance is 2 percent. For cells ddd and eee, the tolerance is 1 percent.

```
LVS CELL LIST L1 aaa bbb ccc
LVS CELL LIST L2 ddd eee
TRACE PROPERTY R R R 4
TRACE PROPERTY R R R 2 CELL LIST L1
TRACE PROPERTY R R R 1 CELL LIST L2
```

**Example 21**

Cancel property tracing of W for MN(NZ) components in list A:

```
TRACE PROPERTY MN(NZ) W W NO CELL LIST A
```

For examples of Trace Property statement that use a *trace\_property\_computation* procedure, see “[Trace Property Computation and Reporting Language](#)” on page 1872.

**Using M property for tracking device counts** — If you want to use the M property to track device counts after reduction, see “[Comparing Device Counts After Reduction](#)” in the *Calibre Verification User’s Manual*.

# Unit Capacitance

Specification statement

**UNIT CAPacitance {*number* | *factor*}**

Used only by LVS and PEX applications.

## Parameters

- ***number***

A required, positive, real number, that specifies the preferred user-unit of capacitance.

- ***factor***

A required unit of capacitance for rule file capacitance equations. This applies to both parasitic and intentional capacitance calculations and reported results. Note that declaring ***factor*** does not specify the preferred unit of capacitance to express results in netlists or reports.

The possible choices are:

<b>aF</b>	Equivalent to 1e-18, an attofarad.
<b>fF</b>	Equivalent to 1e-15, a femtofarad.
<b>pF</b>	Equivalent to 1e-12, a picofarad. This is the default behavior if you do not include this statement in the rule file.
<b>nF</b>	Equivalent to 1e-9, a nanofarad.
<b>uF</b>	Equivalent to 1e-6, a microfarad.
<b>mF</b>	Equivalent to 1e-3, a millifarad.
<b>F</b>	Equivalent to 1, a farad.
<b>kF</b>	Equivalent to 1e3, a kilofarad.
<b>megF</b>	Equivalent to 1e6, a megafarad.
<b>gF</b>	Equivalent to 1e9, a gigafarad.
<b>tF</b>	Equivalent to 1e12, a terafarad.

## Description

Relates the preferred user-unit of capacitance (multiplication factor) to the fixed standard unit of capacitance, which is farads. This statement is optional and, if specified, can appear only once within a rule file. It is independent of any other settings within the process.

If you do not include the Unit Capacitance specification statement within a rule file, then the units for your capacitance equations are, by default, picofarads (pF). Any reported capacitances are consistent with these calculated results. The actual reported units change depending on the magnitude of the results.

**Note**

 The capacitance calculations generated by the xCalibrate rule file generator have a default unit of femtofarads. The Unit Capacitance statement must match the units used in the capacitance calculations or the results will be incorrect.

Unit Capacitance also sets the magnitude of the numeric value returned from the evaluation of a capacitance equation. For example, if a capacitance statement evaluates to 6.3 and you have the following rule file statement:

```
UNIT CAPACITANCE pf
```

then the C value is 6.3 pF. Thus, the value you supply for Unit Capacitance must always agree with the intended units for your capacitance equations.

This statement is not meant to be used as a preferred unit control, such as when you want all C values to be expressed in picofarads in any Calibre output netlist or report. The Unit Capacitance statement only affects calculations and not output.

This statement also controls the output of the UNIT\_CAPacitance() function in the device property computation language.

## Examples

Use picofarads for calculations:

```
UNIT CAPACITANCE pf
```

or

```
UNIT CAPACITANCE 1e-12
```

# Unit Inductance

Specification statement

## UNIT INDUCTANCE {*number* | *factor*}

Used only in Calibre xL.

### Summary

This optional statement specifies the unit of inductance to use for calculating the self inductance of vias and bonding wires.

### Parameters

- *number*

A required, positive, real number specifying the preferred unit of inductance. The choices are:

- 1e-18** Equivalent to aH (attohenries).
- 1e-15** Equivalent to fH (femtohenries).
- 1e-12** Equivalent to pH (picohenries). This is the default behavior if you do not include this statement in the SVRF rule file.
- 1e-9** Equivalent to nH (nanoenries).
- 1e-6** Equivalent to uH (microhenries).
- 1e-3** Equivalent to mH (millihenries).
- 1** Equivalent to H (henry).

- *factor*

A required unit of inductance. The choices are:

- aH** Attohenries; equivalent to 1e-18.
- fH** Femtohenries; equivalent to 1e-15.
- pH** Picohenries; equivalent to 1e-12. This is the default behavior if you do not include this statement in the SVRF rule file.
- nH** Nanoenries; equivalent to 1e-9.
- uH** Microhenries; equivalent to 1e-6.
- mH** Millihenries; equivalent to 1e-3.
- H** Henry; equivalent to 1.
- kH** Kilohenries; equivalent to 1e+3.
- megH** Megahenries; equivalent to 1e+6.

**gH**      Gigahenries; equivalent to 1e+9.

**tH**      Terahenries; equivalent to 1e+12.

## Examples

The examples show the two different ways you can specify the same unit, attohenries, in inductance calculations.

```
UNIT INDUCTANCE aH  
UNIT INDUCTANCE 1e-18
```

## Unit Length

Specification statement

### UNIT LENGTH {*number* | *factor*}

Used only by LVS and PEX applications.

#### Parameters

- *number*

A positive, real number that specifies the preferred user-unit of length.

- *factor*

The user-unit of length. The possible choices are:

<b>u</b>	Equivalent to 1e-6 m, a micron. This is the default behavior for Calibre applications if you do not include this statement in the rule file.
<b>mil</b>	Equivalent to 25.4e-6 m, 1e-3 inch.
<b>mm</b>	Equivalent to 1e-3 m, a millimeter.
<b>cm</b>	Equivalent to 1e-2 m, a centimeter.
<b>inch</b>	Equivalent to 25.4e-3 m.
<b>m</b>	Equivalent to 1 m, a meter.

#### Description

Relates the preferred user-unit of length (multiplication factor) to the fixed standard unit of length, which is meters. It is optional and, if specified, can appear only once within a rule file. In Calibre, the default unit of length is 1 um (1e-6 m).

This statement controls the output of the UNIT\_LENGTH() function in built-in languages for LVS and PEX.

For geometric databases, if the Unit Length setting is inconsistent with the database precision, a warning is issued and the condition is ignored by default. You can alter this behavior by changing the [Layout Input Exception Severity](#) METRIC\_RULE\_FILE setting.

The nmDRC applications do not use the Unit Length statement for any measurements; however, the rule file compiler issues a warning if the Unit Length setting is inconsistent with the layout physical precision as described in the preceding paragraph.

In Pyxis Layout, the factor parameter and the value of the \$unit\_length Process variable do not have to match. If you do not include the Unit Length statement within a rule file, the default unit of length is equal to the value of the \$unit\_length Process Variable.

## Examples

### Example 1

If you include the following Unit Length statement within a rule file, then all numerical values of length in LVS and PEX applications are in microns (um) and reported in scientific notation (1e-6).

```
UNIT LENGTH u
```

### Example 2

The following uses the scientific notation method of writing the above Unit Length statement.

```
UNIT LENGTH 1e-6
```

### Example 3

The following example shows how to set your unit length to one user unit. Often this is interpreted in absolute microns (as opposed to fractions of a meter) for special netlisting purposes. Be aware the SPICE specification calls for lengths to have dimensions of meters, so using the following statement deviates from what SPICE considers valid.

```
UNIT LENGTH 1
```

# Unit Resistance

Specification statement

## UNIT RESistance {*number* | *factor*}

Used only by LVS and PEX applications.

### Parameters

- ***number***

A required, positive, real number that specifies the preferred user-unit of resistance.

- ***factor***

A required user-unit of resistance. The possible choices are:

<b>aohm</b>	Equivalent to 1e-18, an attohm.
<b>fohm</b>	Equivalent to 1e-15, a femtohm.
<b>pohm</b>	Equivalent to 1e-12, a picohm.
<b>nohm</b>	Equivalent to 1e-9, a nanohm.
<b>uohm</b>	Equivalent to 1e-6, a microhm.
<b>mohm</b>	Equivalent to 1e-3, a milliohm.
<b><u>ohm</u></b>	Equivalent to 1, an ohm. This is the default setting.
<b>kohm</b>	Equivalent to 1e3, a kilohm.
<b>megohm</b>	Equivalent to 1e6, a megohm.
<b>gohm</b>	Equivalent to 1e9, a gigohm.
<b>tohm</b>	Equivalent to 1e12, a teraohm.

### Description

Relates the preferred user-unit of resistance to the fixed standard unit of resistance, which is ohms. This statement is optional and, if specified, can appear only once within a rule file. It is independent of any other settings.

If you do not include the Unit Resistance statement within a rule file, the default unit of resistance is 1.

This statement also controls the output of the UNIT\_RESistance() function in the device property computation language.

The nmDRC applications do not use the Unit Resistance statement.

## Examples

If you include the following Unit Resistance statement within a rule file, then all numeric values of resistance are reported in ohms.

```
UNIT RESISTANCE ohm
```

The following is another way to write the previous Unit Resistance statement.

```
UNIT RESISTANCE 1
```

## Unit Time

Specification statement

### UNIT TIME {*number* | *factor*}

Used only by PEX applications.

#### Parameters

- *number*

A required, positive, real number that specifies the preferred user-unit of time.

- *factor*

A required user-unit of time. The possible choices are:

<b>as</b>	Equivalent to 1e-18, an attosecond.
<b>fs</b>	Equivalent to 1e-15, a femtosecond.
<b>ps</b>	Equivalent to 1e-12, a picosecond.
<b>ns</b>	Equivalent to 1e-9, a nanosecond. This is the default setting.
<b>us</b>	Equivalent to 1e-6, a microsecond.
<b>ms</b>	Equivalent to 1e-3, a millisecond.
<b>s</b>	Equivalent to 1, a second.
<b>min</b>	Equivalent to 60 seconds, a minute.
<b>hr</b>	Equivalent to 3600 seconds, an hour.

#### Description

If you do not include the Unit Time statement in a rule file, the default unit of time is 1e-9. This statement affects the D constraint of the [PEX Threshold](#) statement.

#### Examples

If you include the following Unit Time statement in a rule file, then all numerical values of time are reported in nanoseconds.

```
UNIT TIME ns
```

The following is another way to write the above Unit Time statement.

```
UNIT TIME 1e-9
```

# Variable

Specification statement

**VARIABLE** *name* {*value* [*value* ...]} | **ENVIRONMENT**

## Parameters

- ***name***  
A required name of a variable. All ***name*** parameters are case-insensitive.
- ***value***  
A required real number, math expression, or string. More than one *value* is allowed. Strings must be placed inside double quotes.
- **ENVIRONMENT**  
Keyword that specifies that ***name*** is defined in the shell environment. Not used with *value*.

## Description

Specifies to substitute one or more *value* parameters, or a shell-defined variable containing one or more values, for a ***name*** that appears in any layer operation and appropriate specification statements discussed in this section. Variable specification statements are required whenever variables appear in the rule file.

Variables must be declared in Variable statements before the variables are used in the rule file. For example:

```
VARIABLE process ENVIRONMENT
rule { INT metal < process + .05 }
```

These statements are order-dependent because the second statement references a variable called “process”, which must be previously defined.

Variables cannot be redefined once they are declared.

Variable names can appear in other Variable statements where a math expression is used and the value of the first declared variable is dereferenced. For example:

```
VARIABLE width ENVIRONMENT
VARIABLE process_w width * 0.5
```

Here, the width value is dereferenced in the second Variable statement, and 0.5 is multiplied to the value accordingly. However, string-valued Variable names are not dereferenced and concatenated with other string Variable definitions. For example:

```
VARIABLE net1 "a"
VARIABLE nets "b" "net1" // value of nets is "b" "net1" not "b" "a"
```

The value of net1 is not dereferenced in the second Variable statement.

**System variables** — The Calibre system supports three *default variables*. A default variable is a name that has the following characteristics:

- Appears wherever a variable is permitted
- Is not defined in any Variable specification statement, and
- Has a predefined value

The default variables are:

**\$PRECISION** — Resolves to the rule file precision value (which may or may not have been explicitly defined by a [Precision](#) specification statement).

**\$PRECISION\_N** — Resolves to the numerator of the rule file Precision value, if expressed as a ratio; otherwise, resolves to the rule file Precision.

**\$PRECISION\_D** — Resolves to the denominator of the rule file Precision value, if expressed as a ratio; otherwise, resolves to 1.

As with all Variable statement names, these are case-insensitive.

Here is an example:

```
rule { EXT metall1 < 0.025 * $precision }
```

**Permitted uses** — Use of the Variable statement to declare variables is limited to the following cases; no other uses are supported:

- any numeric parameter for any layer operation
- a numeric expression defining a numeric-valued variable in a Variable specification statement
- polygon coordinates in [Polygon](#), [Layout Window](#), or [Layout Windel](#) specification statements
- *tolerance* value in [Trace Property](#) or [LVS Split Gate Ratio](#) specification statements
- filter constraint of an [LVS Filter](#) specification statement
- AREF *width*, or *length*, parameter in a [DRC Check Map](#) statement
- magnification factor in a [Layout Magnify](#), [DRC Magnify Density](#), [DRC Magnify NAR](#), or [DRC Magnify Results](#) statement
- tolerance factor in a [DRC Tolerance Factor](#) statement
- any numeric parameter of any PEX specification statement
- numeric parameters in the parasitic property specification, device property specification, and LVS reduction specification sub-languages
- numeric parameters for the [DRC Map Text Depth](#), [DRC Results Database Precision](#), [Precision](#), [Port Depth](#), and [Text Depth](#) specification statements

- arguments to any of the ENClosure functions of the [Device Property Computation Built-In Language](#)
- as an argument used by a tvf::svrf\_var command.

String variables may be used only in the following cases:

- any cell name parameter of the [Exclude Cell](#), [Expand Cell](#), [Expand Cell Text](#), [Extent Cell](#), [Flatten Cell](#), [Flatten Inside Cell](#), [\(Not\) Inside Cell](#), [Layout Base Cell](#), [Layout Primary\[2\]](#), [Layout Windel Cell](#), [Layout Window Cell](#), [LVS Box](#), [Push Cell](#), and [Source Primary](#) operations and specification statements
- AREF cell name parameter of the [DRC Check Map](#) statement
- the prefix parameter of the DRC Check Map AUTOREF option
- any net name parameter of the [\(Not\) Net](#) operation
- text string parameter of the [Expand Text](#) statement and [\(Not\) With Text](#) operation
- any parameter of the [LVS Component Type Property](#), [LVS Ground Name](#), [LVS Non User Name](#) (INSTANCE, NET, or PORT keywords), [LVS Pin Name Property](#), [LVS Power Name](#), [Virtual Connect Box Name](#), and [Virtual Connect Name](#) specification statements
- the filter constraint in an [LVS Filter](#) statement
- cell names in an [LVS Box](#) statement
- arguments to any of the ENCclosure functions of the [Device Property Computation Built-In Language](#)
- a string argument in a CMACRO statement

String arrays are allowed, as shown in this example:

```
// Select all metal having text beginning with "a", "b", or "c".
VARIABLE metal_text "a?" "b?" "c?"
x = metal WITH TEXT metal_text
```

To see the environment variable implementation, see [Example 3](#).

**Variables in comments** — Variables can be resolved inside of rule check (@) comments to enhance DRC results presentation. This is done by placing a caret (^) in front of the variable name. To escape the ^, precede it with a backslash (\). For example, assume that METAL\_SPACING is a variable with the value 2. Then the rule check:

```
Rule_6.4 {
@ Metal spacing must be ^METAL_SPACING microns.
EXTERNAL metal < METAL_SPACING }
```

actually compiles to:

```
Rule_6.4 {  
@ Metal spacing must be 2 microns.  
EXTERNAL metal < 2 }
```

**Restrictions** — The Variable statement cannot be used for any of these purposes:

- in place of SVRF keywords (The primary keyword list that causes conflicts is in [Table 2-4](#). Secondary keywords may be allowed as Variable names, but this is better off avoided as these can cause problems in some cases.)
- file name (an [Include](#) statement embedded in a Variable statement is an exception), layer name, layer number, DATATYPE, or TEXTTYPE parameters
- element name, model name, pin name, or property name parameters in a [Device](#) statement
- LINK name in an [Sconnect](#) operation
- rule check or [Group](#) names
- names in pre-processor (#IFDEF) directives
- name parameters in PEX family, [Title](#), or
- family of specification statements, except where specifically mentioned previously in this section
- numeric parameters in any of the nmDRC family, ERC family, [Layer Resolution](#), [Layout Bump2](#), [Layout Property Text](#), [Port Depth](#), [Precision](#), [Resolution](#), [Text Depth](#), [Text Print Maximum](#), or the Unit family of specification statements
- name or numeric parameters in the LVS family of specification statements, except where specifically mentioned previously in this section
- SPICE values in [LVS Filter](#), [LVS Split Gate Ratio](#), or [Trace Property](#) specification statements
- text name or location parameters in [Layout Rename Text](#), [Text](#), or [Layout Text](#) specification statement

---

**Note**

In many cases, the limitations of the Variable statement can be overcome by using a TVF rule file with Tcl variables.

---

**ICverify considerations** — In ICverify, variables can be resolved in the Pyxis Layout Process. If a variable is defined both in the Pyxis Layout Process and in the rule file via a Variable specification statement, then the definitions must match. That is, in ICverify, Variable specification statements cannot redefine variable names defined in the Pyxis Layout Process. An Pyxis Layout Process variable can also be used to define a variable in a Variable specification statement. If the same variable name is defined in the Process, then the values

must match. Variable resolution occurs at rule file compilation time. If the value is changed while the rule file is loaded, then that value will not represent the true state of Pyxis Layout.

## Examples

### Example 1

In the following example, alpha is defined in terms of variable grid. Variable X can be defined in terms of variable Y only if the definition of Y *precedes* that of X in the rule file. This is one of a few cases where order dependency affects an SVRF rule file. Variables cannot be redefined.

```
VARIABLE grid 4
VARIABLE alpha ( grid - 1 ) * 18.2 / 6
...
R8.4 { EXT metal < alpha + 2 }
```

### Example 2

In the following example, POLY\_SPACING is defined in the C shell environment as follows:

```
setenv POLY_SPACING "7"
```

and the following Variable statement specifies that the value of POLY\_SPACING was defined in the shell environment:

```
VARIABLE POLY_SPACING ENVIRONMENT
```

### Example 3

The question mark (?) wildcard for the With Text operation matches 0 or more characters in a string. This can also be used in variable declarations.

```
// Select all metal having text beginning with "a", "b", or "c".
VARIABLE metal_text "a??" "b??" "c??"
x = metal WITH TEXT metal_text
```

To use an environment variable, set this in your shell:

```
setenv metal_text '"a??" "b??" "c??"'
or
metal_text='a??" "b??" "c??"'; export metal_text
```

Then do the following in a compile-time TVF rule file:

```
#! tvf
if [ info exists env(metal_text) ] {
    tvf::VERBATIM "VARIABLE metal_text $env(metal_text)"
}
```

# Vertex

Layer operation

**VERTEX layer constraint****Parameters**

- *layer*

A required original layer or layer set, or a derived polygon layer.

- *constraint*

A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

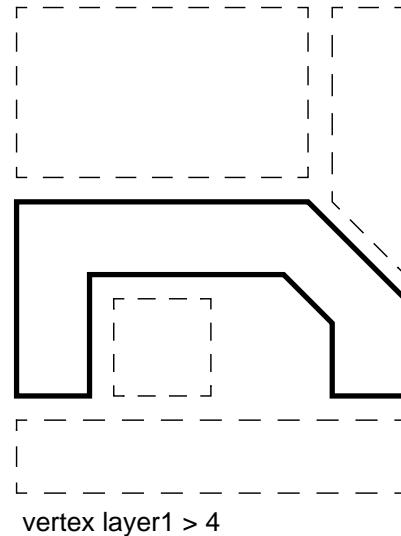
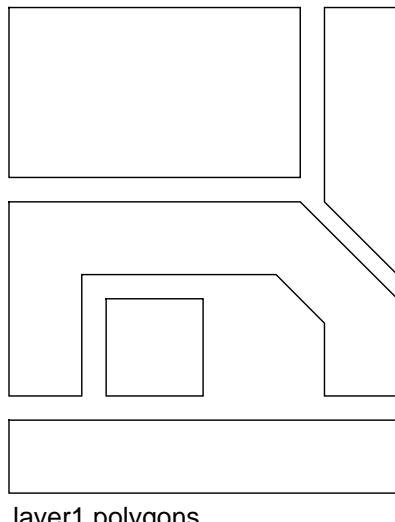
The constraint must contain non-negative integers.

**Description**

Selects all *layer* polygons having vertex counts conforming to the constraint. Note that for any polygon, the number of edges matches the number of unique vertices, so this operation can be used for edge counting.

**Examples****Example 1**

The Vertex operation shown in Figure 4-337 selects all layer1 polygons that have a vertex count greater than 4.

**Figure 4-337. Vertex****Example 2**

```
// Generate a clone of the metal layer with all polygons except for those
// having more than 1024 vertices.
fewer_metal_vertices = VERTEX <= 1024 metal
```

# Virtual Connect Box Colon

Specification statement

## VIRTUAL CONNECT BOX COLON {NO | YES}

Used only in Calibre nmLVS-H and Calibre xRC.

### Parameters

- **NO**

Keyword that does not cause virtual connections for net names containing a colon or a semicolon character in [LVS Box](#) cells. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that causes virtual connections for identical net names containing a colon character or semicolon character in LVS Box cells.

### Description

Specifies virtual connections for identical net names containing the colon (:) or semicolon (;) characters in [LVS Box](#) cells. Virtual Connect Box Colon operates on box cells at all levels of hierarchy and it applies to the layout as specified in the LVS Box statement. You can specify the Virtual Connect Box Colon statement once. The behavior is similar to the [Virtual Connect Colon](#) specification statement.

The YES keyword specifies to remove colon suffixes from labels in LVS Box cells, and then virtually connects nets with identical labels up to the discarded suffixes. Labels that do not have colons in them are not affected by this statement. It does not remove colon suffixes from node names in .GLOBAL statements in SPICE netlists, which is different from the Virtual Connect Colon statement.

Virtually connected pins act as feedthroughs and connect nets that are attached to them at higher levels of hierarchy.

Note that the [LVS Isolate Shorts](#) specification statement does not process virtual box cell connections. Thus, the short isolator does not find feedthrough paths created by this statement. The connectivity extractor reports a short circuit warning, but the short isolator cannot isolate the short.

When allowing virtual connections, it is recommended to specify [Virtual Connect Report YES](#).

See also [Virtual Connect Box Name](#) and [Virtual Connect Semicolon As Colon](#).

### Examples

#### Example 1

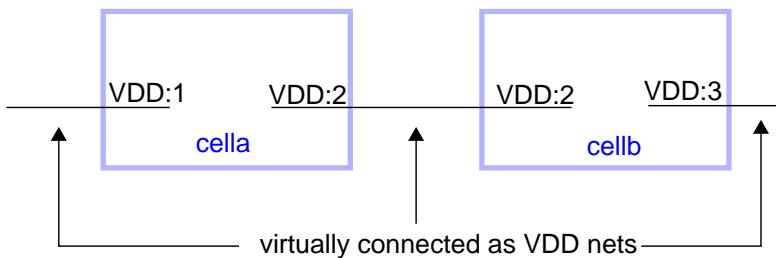
```
VIRTUAL CONNECT BOX COLON NO
// do not virtually connect identical net names with colon suffixes in
// LVS Box cells.
```

**Example 2**

Assume the following:

```
LVS BOX cella cellb  
VIRTUAL CONNECT BOX COLON YES
```

**Figure 4-338. Virtual Connect Box Colon**



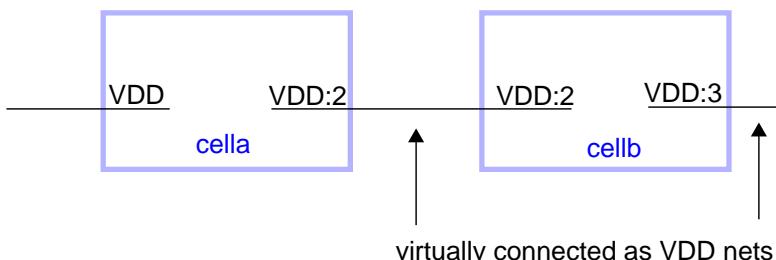
Net names VDD:1, VDD:2, and VDD:3 are virtually connected with name VDD in cella and cellb.

**Example 3**

Assume the following:

```
LVS BOX cella cellb  
VIRTUAL CONNECT BOX COLON YES
```

**Figure 4-339. Virtual Connect Box Colon**



Net labels VDD and VDD:n are not virtually connected. If each of these labels appears on physically distinct nets, a warning is issued about two nets having the same name, and VDD is assigned to one of the nets arbitrarily. However, if both labels are on the same physical net, and there are no other conflicts, then that net is assigned the name VDD.

# Virtual Connect Box Name

Specification statement

**VIRTUAL CONNECT BOX NAME** *name* [*name* ...]

Used only in Calibre nmLVS-H and Calibre xRC.

## Parameters

- *name*

A required net name. You can specify *name* any number of times in one statement. The *name* can be a string variable. The question mark (?) is a wildcard that identifies identical patterns of zero or more characters.

## Description

Specifies virtual connections for identical (case insensitive) net names in [LVS Box](#) cells. Virtual Connect Box Name statement operates on box cells at all levels of hierarchy and it applies to the layout as specified in the LVS Box statement. You can specify the Virtual Connect Box Name statement any number of times. The behavior is similar to the [Virtual Connect Name](#) specification statement.

Several different *name* parameters can be listed in the same statement, but different net names are never connected together. For example:

```
VIRTUAL CONNECT BOX NAME "VDD" "GND"
```

This example causes virtual connections between physically disjoint nets named VDD. It also virtually connects physically disjoint nets named GND. However, it does not cause virtual connections between nets named VDD and GND.

The ? wildcard is useful for virtually connecting many identically-named nets using a single *name* parameter in a manner similar to the preceding paragraph. For example, “VDD?” virtually connects physically disjoint nets VDD to VDD and VDD1 to VDD1, but it does not virtually connect VDD to VDD1.

Virtually connected pins act as feedthroughs and connect nets that are attached to the pins at higher levels of hierarchy.

Note that the [LVS Isolate Shorts](#) specification statement does not process virtual box cell connections. Thus, the short isolator will not find feedthrough paths created by this statement. The connectivity extractor reports a short circuit warning, but the short isolator cannot isolate the short.

When allowing virtual connections, it is recommended to specify [Virtual Connect Report YES](#).

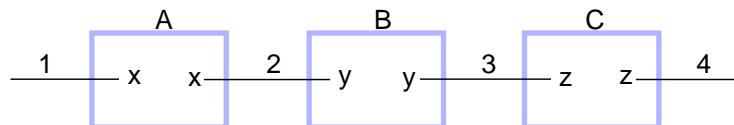
See also [Virtual Connect Box Colon](#).

## Examples

### Example 1

In [Figure 4-340](#) the nets marked 1, 2, 3, and 4 all belong to the same net because the bare ? wildcard matches identical net names in box cells.

**Figure 4-340. Virtual Connect Box Name**



LVS BOX "A" "B" "C"  
VIRTUAL CONNECT BOX NAME "?"

### Example 2

The following example connects net segments with identical names in box cells, provided that the names are either VCC or VSS (case is ignored). It also connects their respective pins.

VIRTUAL CONNECT BOX NAME "VCC" "VSS"

# Virtual Connect Colon

Specification statement

## VIRTUAL CONNECT COLON {NO | YES}

### Parameters

- **NO**

Keyword that does not cause virtual connections for net names containing a colon or a semicolon character. This is the default behavior when you do not include this statement in the rule file.

- **YES**

Keyword that creates virtual connections for identical net names up to the appearance of a colon or a semicolon character.

### Description

Specifies virtual connections for identical net names containing the colon (:) or semicolon (;) character. This applies to both layout and source net names.

Net labels can be added in the layout using [Text](#) or [Layout Text](#) specification statements, or using geometric database text. A virtual connection between labels causes a virtual connection between the net segments to which those labels are assigned, regardless of whether or not the label name becomes the final name of the net segment.

If YES is specified, then the connectivity extractor forms a virtual connection between any two labels that have the same name up to the first appearance of a colon. That is, the colon and everything after it are not considered when the extractor attempts to match net names. Colons can appear anywhere in the name with the exception that a colon at the beginning of a name is treated as a regular character (that is, it has no special effect). For example, VCC:1, VCC:10, and VCC: would be virtually connected if YES is specified. This behavior is also true for node names in .GLOBAL statements in SPICE netlists when you specify YES.

Virtual Connect Colon YES also removes colon suffixes from names of port objects in the layout in addition to net labels. The removal is done during connectivity extraction and is performed in the same manner as in net labels. It is performed in both hierarchical and flat connectivity extraction and is done at all levels of hierarchy. However, port objects do not form virtual connections; you have to use net labels to do that. Named port objects are created with the [Port Layer Text](#) specification statement or, in Pyxis Layout, with the \$make\_port() function.

If NO is specified (the default), then there is no special treatment of colon characters in label names or port names. In particular, colon suffixes are not stripped off and no virtual connections are performed based on the presence of colon characters.

This statement is primarily intended for LVS applications.

When allowing virtual connections, it is recommended to specify [Virtual Connect Report](#) YES.

You can specify virtual connect behavior in Calibre Interactive from the Connect tab in the Setup > Options pane. See the [Calibre Interactive and Calibre RVE User's Manual](#) for details.

## Virtual Connect Colon

In Pyxis Layout, net labels can also be entered with free-standing text and Pyxis Layout net properties, in addition to the methods described previously in this section.

See also [Virtual Connect Name](#), [Virtual Connect Depth](#), [Virtual Connect Incremental](#), [Virtual Connect Semicolon As Colon](#), [Virtual Connect Box Colon](#), and [Virtual Connect Report](#).

## Examples

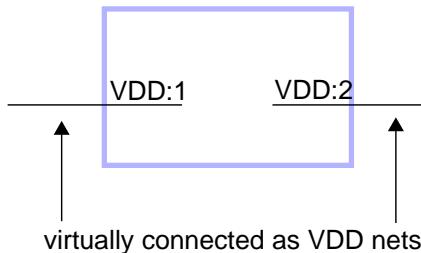
### Example 1

```
VIRTUAL CONNECT COLON NO // default; no virtual connections for net names  
// with colon or semicolon characters
```

### Example 2

```
VIRTUAL CONNECT COLON YES // matching net labels after removal of suffixes  
// are virtually connected
```

**Figure 4-341. Virtual Connect Colon YES**



### Example 3

Given:

```
VIRTUAL CONNECT COLON YES
```

During connectivity extraction:

- Two disjoint net segments labeled VDD: and VDD: are virtually connected. The resulting net name is VDD.
- Three disjoint net segments labeled BAR:, BAR:X and BAR:ABC are virtually connected. The resulting net name is BAR.
- Two disjoint net segments labeled XYZ and XYZ: are NOT virtually connected by colons because XYZ does not contain a colon. You get a warning about two nets with the same name and the name XYZ is assigned to one of them, chosen arbitrarily.
- Three disjoint net segments labeled XYZ, XYZ: and XYZ: are not all virtually connected. However, the two net segments labeled XYZ: are virtually connected. You get a warning about two nets with the same name, and the name XYZ is assigned to one of them arbitrarily.

- Three labels, XYZ, XYZ:, and XYZ:ABC, on one physical net become the net named XYZ.
- A net labeled XYZ: in the layout and XYZ in the schematic become an initial correspondence point in LVS because the colon suffix in the layout is stripped off.
- A net labeled XYZ:ABC in the layout is named XYZ when you generate a SPICE netlist with layout names.
- .GLOBAL VCC:P VSS:XYZ VDD: becomes .GLOBAL VCC VSS VDD.

# Virtual Connect Depth

Specification statement

## VIRTUAL CONNECT DEPTH {PRIMARY | ALL}

Used only in Calibre nmLVS-H and Calibre xRC.

### Parameters

- **PRIMARY**

Required keyword that specifies virtual connections should be performed only in the top-level cell. This is the default behavior.

- **ALL**

Required keyword that specifies virtual connections should be performed at all levels of hierarchy.

### Description

Specifies whether virtual connections should be performed only in the top-level cell or at all levels of hierarchy. This statement may be specified once.

When PRIMARY is specified, virtual connections are performed only in the top-level cell using text in the top-level cell. Text used in the top-level cell is subject to the [Text Depth](#) statement.

When ALL is specified, virtual connections are performed in all cells at all levels of hierarchy using text local to each cell. For flat applications, ALL is equivalent to PRIMARY.

---

#### **Caution**



The ALL setting should be used with care since text at lower levels of hierarchy may not be reliable. As a result, layout open circuits could be missed. Virtual connections should be disabled altogether when verifying a complete design.

---

Virtual connections are enabled with [Virtual Connect Colon](#) and [Virtual Connect Name](#) specification statements. The Virtual Connect Depth statement has no effect unless virtual connections are enabled.

Additional details—

- Feedthrough pins

When ALL is specified, virtually-connected pins of child cells act as feedthroughs and connect nets that are connected to those pins at higher levels of hierarchy.

- [LVS Box](#) cells

When ALL is specified, regular virtual connections, indicated with Virtual Connect Colon and Virtual Connect Name specification statements, are performed in LVS Box cells *in addition to* box-specific virtual connections indicated with [Virtual Connect Box Colon](#) and [Virtual Connect Box Name](#) specification statements. Note that the box-

specific statements always operate at all levels of hierarchy, regardless of the Virtual Connect Depth setting.

- Colon stripping

Stripping of colon suffixes as directed by the Virtual Connect Colon specification statement is always done at all levels of hierarchy, regardless of the Virtual Connect Depth setting.

- Virtual connect reporting

When reporting of virtual connections is requested, virtual connections are reported as directed by the Virtual Connect Depth setting. (Virtual connection reporting is requested with [Virtual Connect Report YES](#) or [LVS Report Option V](#)). Virtual connections are performed after physical connectivity has been established at all levels of hierarchy.

When enabling virtual connections, it is recommended to specify [Virtual Connect Report YES](#).

## Examples

```
//DISABLE FOR FINAL VERIFICATION

VIRTUAL CONNECT NAME "VDD" "VSS"
VIRTUAL CONNECT COLON YES
VIRTUAL CONNECT DEPTH PRIMARY
VIRTUAL CONNECT REPORT YES
```

## Virtual Connect Incremental

Specification statement

### VIRTUAL CONNECT INCREMENTAL { NO | YES }

Used only in Calibre nmDRC-H.

#### Parameters

- **NO**

Keyword that instructs Calibre not to apply virtual connection within all [Connect](#) blocks in an incremental connectivity run. This is the default behavior if you do not include this statement in your rule file.

- **YES**

Keyword that instructs Calibre to apply virtual connection within all Connect blocks in an incremental connectivity run.

#### Description

Allows incremental virtual connections in Calibre nmDRC-H (hierarchical) flows. If Virtual Connect Incremental YES is specified, then virtual connection is applied within all Connect blocks.

By default, virtual connection in Calibre nmDRC-H with [DRC Incremental Connect](#) YES is only performed within the last Connect block or within a Connect block where there is a [Net](#) or [Not Net](#) operation between the current Connect block and the subsequent one.

This statement does not apply to Calibre nmDRC (flat) since virtual connection is always applied within all Connect blocks in a flat incremental flow. However, virtual connect warnings are only emitted in the cases discussed in the preceding paragraph.

See also [Virtual Connect Colon](#) and [Virtual Connect Name](#).

#### Examples

##### Example 1

This example shows the default behavior.

```
VIRTUAL CONNECT NAME vdd?  
VIRTUAL CONNECT INCREMENTAL NO  
  
DRC INCREMENTAL CONNECT YES  
  
CONNECT gate poly  
//drc operations  
//virtual connect not applied here  
CONNECT poly metall1 by contact  
//drc operations  
X = NET metall1 vdda  
//virtual connect applied here  
CONNECT metall1 metall2 BY vial
```

```
...
//virtual connect not applied where [NOT] NET is not used
CONNECT metal9 metal10 BY via9
//final connect zone
//virtual connect applied here
```

### Example 2

This example shows virtual connectivity applied globally with incremental connectivity.

```
VIRTUAL CONNECT NAME vdd?
VIRTUAL CONNECT INCREMENTAL YES

//virtual connections applied in all connect zones
DRC INCREMENTAL CONNECT YES
```

## Virtual Connect Name

Specification statement

**VIRTUAL CONNECT NAME** *name* [*name* ...]

### Parameters

- *name*

A required net name. You can specify *name* any number of times in one statement. The *name* can be a string variable. The question mark (?) is a wildcard that identifies identical patterns of zero or more characters.

### Description

Specifies virtual connections for identical (case insensitive) net names. Any set of disjoint nets that share the same name become a single net, if that name appears in a Virtual Connect Name statement. This statement can be specified any number of times.

Several different *name* parameters can be listed in the same statement, but different net names are never connected together. For example:

```
VIRTUAL CONNECT NAME "VDD" "GND"
```

This statement causes virtual connections between physically disjoint nets named VDD. It also virtually connects physically disjoint nets named GND. However, it does not cause virtual connections between nets named VDD and GND.

The ? wildcard is useful for virtually connecting many identically-named nets using a single *name* parameter in a manner similar to the preceding paragraph. For example, “VDD?” virtually connects physically disjoint nets VDD to VDD and VDD1 to VDD1, but it does not virtually connect VDD to VDD1.

If [Virtual Connect Colon](#) is also specified, then Virtual Connect Name operates on names after all colon suffixes have been removed.

The hierarchical depth to which virtual connections are made is controlled by the [Virtual Connect Depth](#) statement.

This statement is primarily intended for LVS applications.

When allowing virtual connections, it is recommended to specify [Virtual Connect Report YES](#).

You can specify virtual connect behavior in Calibre Interactive from the Connect tab in the GUI Options pane. See the [Calibre Interactive and Calibre RVE User's Manual](#) for details.

See also [Virtual Connect Box Name](#), [Virtual Connect Incremental](#), and [Virtual Connect Report](#).

### Examples

#### Example 1

The following example specifies that any set of disjoint nets with the name VDD will become a single net. Also, any set of disjoint nets that share the name VSS will become a single net.

```
VIRTUAL CONNECT NAME "VDD" "VSS"
```

**Example 2**

In the following example, suppose you have four disjoint net segments labeled VSS, VSS:, and VSS:X. The Virtual Connect Colon statement strips off the colon suffixes. The four nets are then virtually connected as one net named VSS.

```
VIRTUAL CONNECT COLON YES  
VIRTUAL CONNECT NAME "VSS"
```

**Example 3**

In the following example, suppose you have four disjoint net segments labeled VSS, VSS:, and VSS:X. The two net segments labeled VSS are virtually connected, but the nets labeled VSS: and VSS:X remain separate nets.

```
VIRTUAL CONNECT COLON NO  
VIRTUAL CONNECT NAME "VSS"
```

**Example 4**

The following statement connects any two nodes having labels with the same name and beginning with the characters VEE or VSS. It connects VEE1 to VEE1, VEE2 to VEE2, VSSXX to VSSXX, and so forth. Note that it will not connect different names; for example it will not connect VEE1 to VEE2.

```
VIRTUAL CONNECT NAME "VEE?" "VSS?"
```

**Example 5**

The following statement connects together any two labels that have the same name:

```
VIRTUAL CONNECT NAME "?"
```

# Virtual Connect Report

Specification statement

## VIRTUAL CONNECT REPORT {NO | YES [UNSATISFIED]}

### Parameters

- **NO**  
Keyword that instructs the tool not to report warnings or notes when a virtual connection is made. This is the default behavior when you do not include this statement in the rule file.
- **YES**  
Keyword that instructs the tool to report warnings or notes when a virtual connection is made.
- **UNSATISFIED**  
Optional keyword that specifies to report virtual connections between net labels that are not connected physically in the super-hierarchy of a cell (that is, the hierarchy above where the cell placements occur), when some placements of the cell do not have any such physical connections in the super-hierarchy. This keyword may only be specified with the YES keyword.

### Description

Instructs Calibre applications whether to report when a virtual connection is made by the connectivity extractor. For non-nmDRC applications, specifying YES is equivalent to specifying [LVS Report Option V](#).

When YES is specified, for each virtual connection made, a WARNING (nmDRC applications) or a NOTE is issued. The message consists of the text label name and two coordinates. Virtual connections between physically connected labels are neither performed nor reported.

You can specify virtual connect behavior in Calibre Interactive from the Connect tab in the GUI Options pane. See the [Calibre Interactive and Calibre RVE User's Manual](#) for details.

### Hierarchical Considerations

In hierarchical applications, virtual connections are reported (when requested by an appropriate Virtual Connect specification statement) from all levels of the hierarchy where such connections occur, subject to the [Virtual Connect Depth](#) statement setting.

When YES is specified, the following behaviors hold:

- Virtual connections are performed bottom-up. That is, virtual connections in a parent cell are performed after virtual connections have been completed in all of its children. Thus, virtual connections between labels in a parent cell are neither performed nor reported if those labels are virtually connected at lower levels of the hierarchy.
- If UNSATISFIED is not specified, virtual connections are performed before high short resolution. Thus, virtual connections in lower-level cells are reported even if high short resolution would have connected the respective nets.

When the UNSATISFIED keyword is specified, physical connections are checked in the super-hierarchy of a cell. Such connections may be present in some or all placements of a cell. If such connections occur in all placements of a cell, then virtual connections are not reported for that cell.

When virtually connected net labels are not physically connected in the super-hierarchy of *all placements* of a cell, then the virtual connections are reported with WARNING messages for those placements that lack the physical connection in their super-hierarchy. Virtual connections are only reported at the highest level of the hierarchy in which both labels appear.

To limit the amount of data being reported when the UNSATISFIED keyword is used, only one text label is reported per virtual connection per physical net. When a physical net has more than one label with the same name, one of the labels is chosen arbitrarily. For each group of net labels that participate in a virtual connection, the following information is reported:

- Label — The virtually connected label.
- Cell name — Name of the cell that contains the labels.
- Placement — Hierarchical pathname of the placement where the virtual connection occurred starting at the top-level cell.
- Placement extent — Coordinates of the lower-left and upper-right corners of the bounding box of the placement where the virtual connection occurred, in top-level coordinate space.
- Net ID — Hierarchical net path from the top-level cell, including the net number of the net to which the labels are attached. This net number represents final connectivity after virtual connections are performed.
- For each net label that participates in the virtual connection, the following are reported:
  - Label location in top-level coordinate space.
  - Label layer number and name (if applicable).

For example:

```
Extraction Errors and Warnings for cell "chip"
-----
```

```
WARNING: Virtually connected Label "D#2" in "cellA"
Placement: "X24/X3/X17" (-20,-30) (-10,-15)
Net Id: X24/X3/X17/25
(1) "D#2" at location (-16,-19) on layer 4 "M1"
(2) "D#2" at location (-16,-20) on layer 4 "M1"
```

You can limit the number of report messages by using the [Virtual Connect Report Maximum](#) specification statement.

See also [Virtual Connect Colon](#), [Virtual Connect Name](#), [Virtual Connect Box Colon](#), [Virtual Connect Box Name](#), and [Virtual Connect Semicolon As Colon](#).

## Examples

### Example 1

This example shows virtual connections being enabled for labels starting with VDD.

```
VIRTUAL CONNECT NAME "VDD?"  
VIRTUAL CONNECT REPORT YES  
VIRTUAL CONNECT DEPTH PRIMARY
```

The nets in [Figure 4-342](#) would be reported.

**Figure 4-342. Virtual Connections**



VDD nets virtually connected

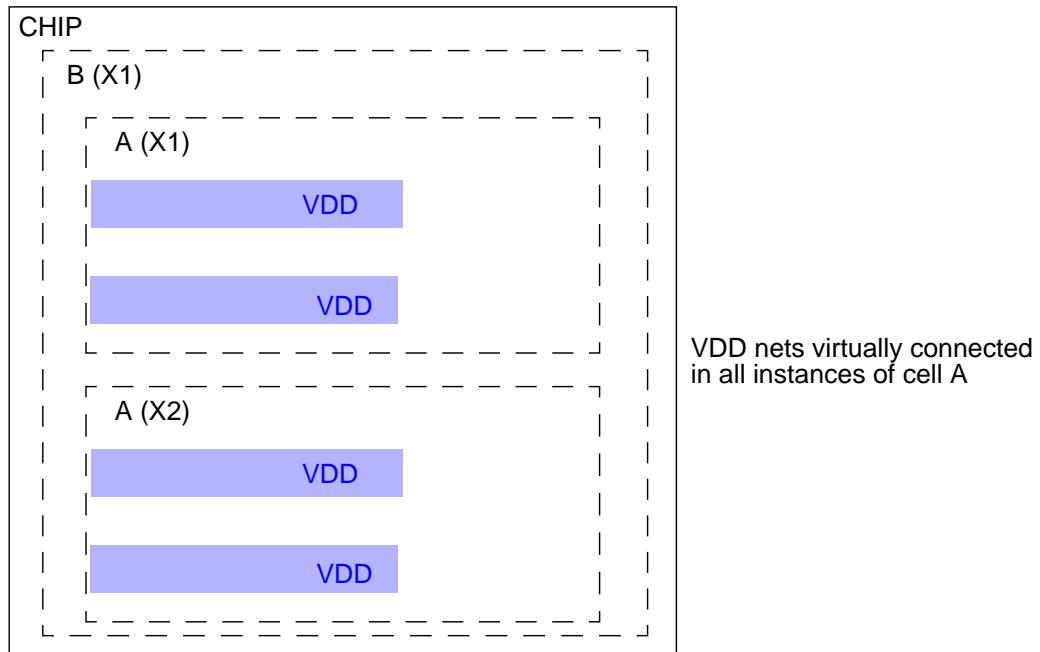
**UNSATISFIED Keyword Examples**

For each of the following examples, assume the following are specified:

```
VIRTUAL CONNECT REPORT YES UNSATISFIED
VIRTUAL CONNECT DEPTH ALL
VIRTUAL CONNECT NAME "VDD?"
```

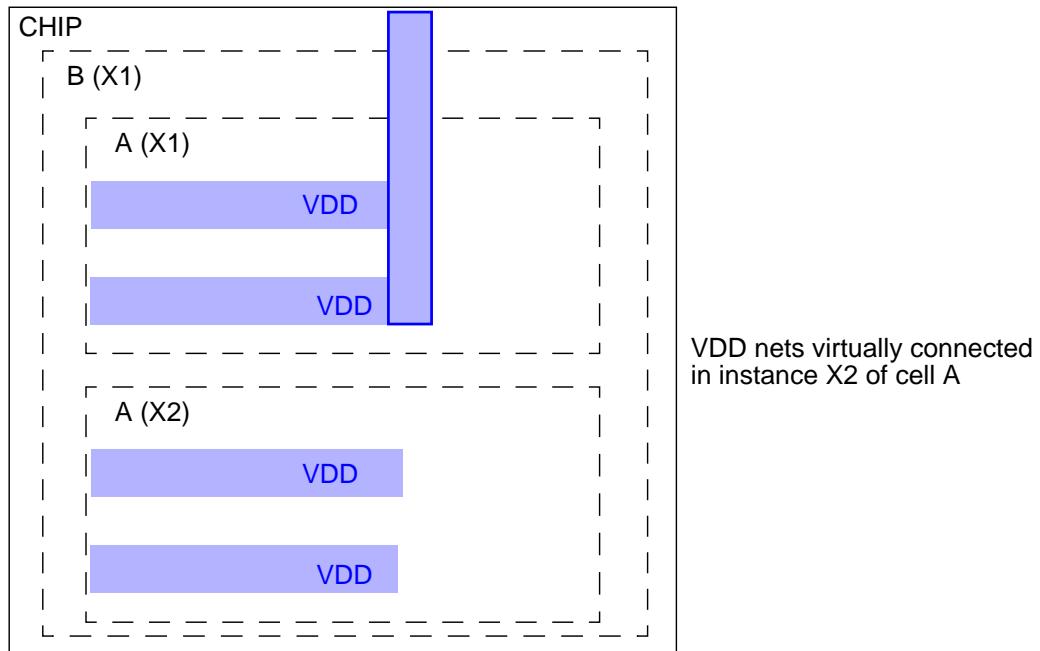
**Example 2**

This example shows two placements of cell A within cell B, which in turn is placed in cell CHIP. Cell A contains two nets labeled VDD. A virtual connection is reported for each of the placements of cell A. The placements are identified as X1/X1 and X1/X2.



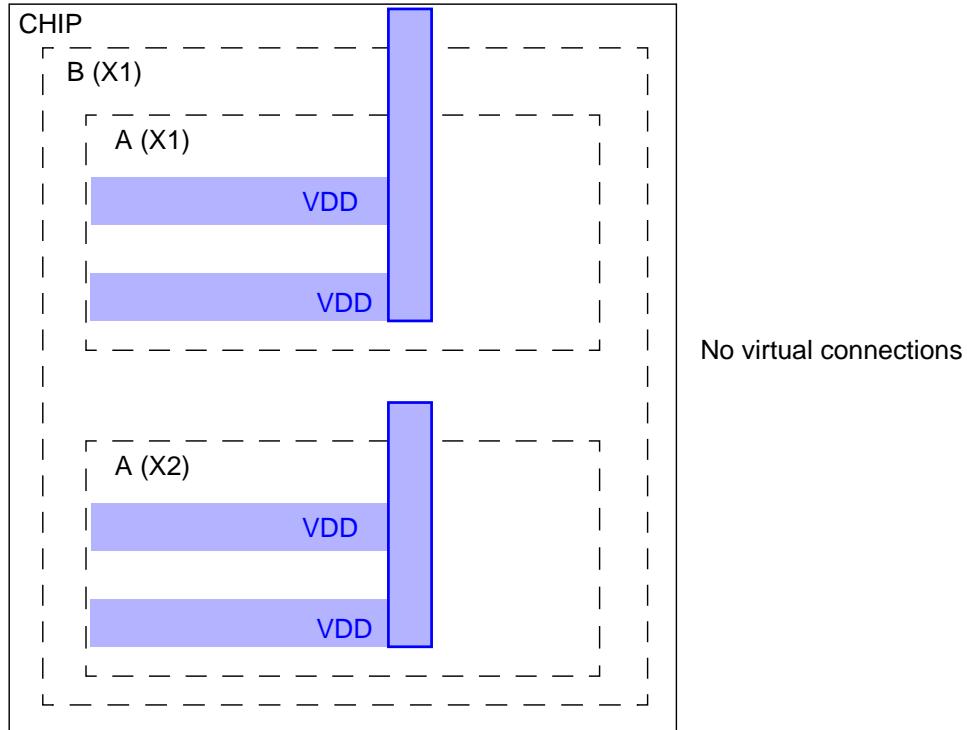
**Example 3**

This example shows two placements of cell A within cell B, which in turn is placed in cell CHIP. Cell A contains two nets labeled VDD. There is a polygon in cell CHIP connecting the two nets VDD in the X1 instance of cell A. A virtual connection is reported for the X2 instance of cell A, but not for the X1 instance. The X2 instance is identified as X1/X2.



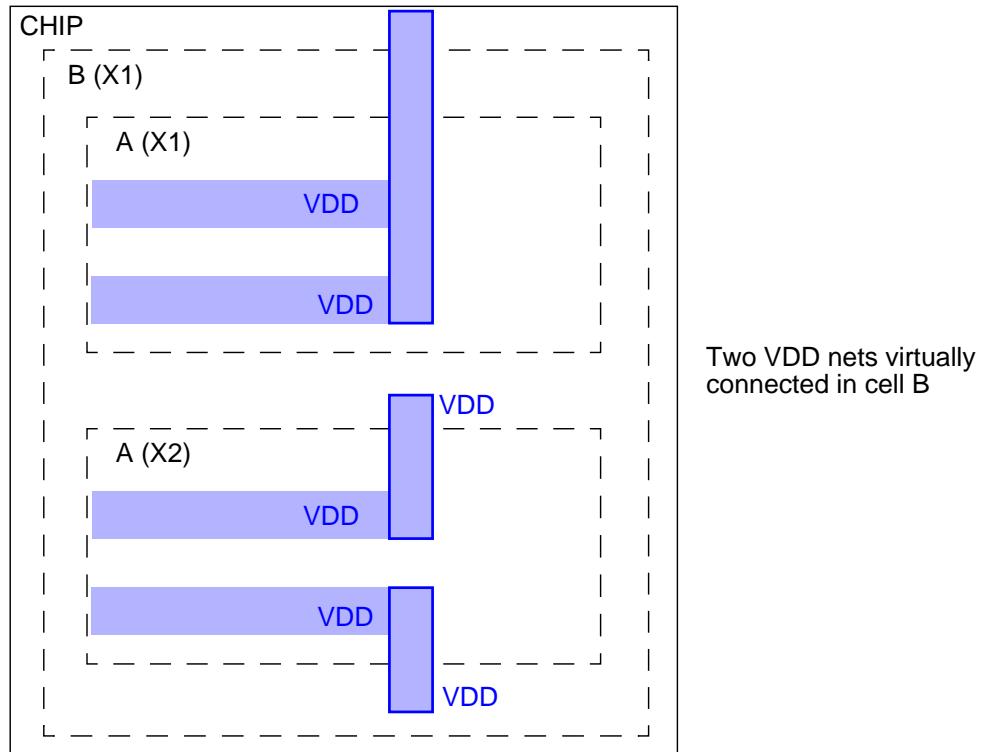
**Example 4**

This example shows two placements of cell A within cell B, which in turn is placed in cell CHIP. Cell A contains two nets labeled VDD. There is a polygon in cell CHIP connecting the two nets VDD in the X1 instance of A. There is a polygon in cell B connecting the two nets VDD in the X2 instance of A. No virtual connections are reported because the VDD nets are physically connected in the super-hierarchy in all placements of A.



**Example 5**

This example shows two placements of cell A within cell B, which in turn is placed in cell CHIP. Cell A contains two nets labeled VDD. There is a polygon in cell CHIP connecting the two nets VDD in instance X1 of A. There are additional VDD texted polygons in cell B that connect to the VDD nets in instance X2 of cell A. Only the virtual connection in placement X1 of cell B is reported in this case.



# Virtual Connect Report Maximum

Specification statement

## VIRTUAL CONNECT REPORT MAXIMUM {ALL | *number*}

### Parameters

- **ALL**

Keyword that specifies to report all virtual connections. This is the default if this statement is not included in the rule file.

- ***number***

Positive integer that specifies the maximum number of virtual connect messages reported during the run.

### Description

Instructs the Calibre tool to limit the number of virtual connect messages reported during the run.

This statement affects the number of NOTE or WARNING (in nmDRC) messages issued when [Virtual Connect Report YES](#) or [LVS Report Option V](#) is specified. This statement also affects the number of WARNING messages when the [Virtual Connect Report YES UNSATISFIED](#) is specified.

See also [Virtual Connect Colon](#), [Virtual Connect Name](#), [Virtual Connect Box Colon](#), [Virtual Connect Box Name](#), [Virtual Connect Depth](#), and [Virtual Connect Semicolon As Colon](#).

### Examples

```
VIRTUAL CONNECT REPORT YES
// Report only the first 99 virtual connections
VIRTUAL CONNECT REPORT MAXIMUM 99
```

## Virtual Connect Semicolon As Colon

Specification statement

**VIRTUAL CONNECT SEMICOLON AS COLON {YES | NO}**

### Parameters

- **YES**

Keyword that causes virtual connections for net names containing a semicolon character (;). This is the default behavior; however, activation of this statement depends on the settings of Virtual Connect Colon and Virtual Connect Box Colon statements.

- **NO**

Keyword that instructs the tool not to virtually connect net names with semicolon characters.

### Description

Specifies whether “;” characters should behave as colons (:) with respect to the [Virtual Connect Colon](#) and [Virtual Connect Box Colon](#) specification statements. Note that these two statements default to NO.

When allowing virtual connections, it is recommended to specify [Virtual Connect Report YES](#).

### Examples

```
/* virtually connect nets having identical net names up to colon or
semicolon characters */
VIRTUAL CONNECT COLON YES
VIRTUAL CONNECT SEMICOLON AS COLON YES
```

## With Edge

Layer operation

**WITH EDGE** *layer1 layer2 [constraint]*

### Parameters

- *layer1*  
A required original layer or layer set, or a derived polygon layer.
- *layer2*  
A required derived edge layer.
- *constraint*  
An optional constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. Specifies the number of edges or edge segments a *layer1* polygon must have on *layer2* to be selected by the With Edge operation.

### Description

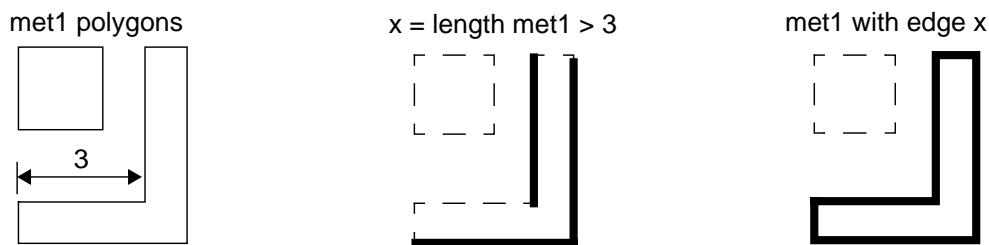
Selects all *layer1* polygons that have edges or edge segments on *layer2*. The *constraint* limits selection of polygons from *layer1* according to the number of edges or segments that the polygon has in *layer2*. See also [Not With Edge](#).

### Examples

#### Example 1

The With Edge operation shown in Figure 4-343 selects all met1 polygons that have edges longer than three units.

**Figure 4-343. With Edge**



#### Example 2

Find all metal polygons having at least one edge longer than 100 microns.

```
x = metal LENGTH > 100
long_metal_polygons = metal WITH EDGE x
```

#### Example 3

Find all metal polygons having a width less than 3 microns.

```
x = INTERNAL [metal] < 3
thin_metal_polygons = metal WITH EDGE x
```

## With Edge

---

### Example 4

Find all contacts that have inside coincident edges with metal.

```
y = contact COINCIDENT INSIDE EDGE metal  
contact_touch_inside = contact WITH EDGE y
```

### Example 5

Find contacts with exactly two edges on layer y.

```
z = contact WITH EDGE y == 2
```

## With Neighbor

Layer operation

**WITH NEIGHBOR *layer1* [*layer2*] *constraint1* SPACE *constraint2***  
 [SQUARE] [CENTERS [OCTAGONAL ONLY | ORTHOGONAL ONLY]]  
 {[INSIDE OF LAYER *layer3*] | [[NOT] CONNECTED]}

### Summary

Selects orthogonal (with respect to the database axes) rectangles if they have the specified number of orthogonal rectangles within the specified distance.

### Parameters

- ***layer1***  
 A required original layer or layer set, or a derived polygon layer.
- ***layer2***  
 An optional original layer or layer set, or a derived polygon layer.
- ***constraint1***  
 A required constraint listed in the “Constraint Notation” column of [Table 2-2](#) on page 45. The constraint must contain non-negative integers. Specifies the number of orthogonal (with respect to the database axes) rectangles a given orthogonal rectangle must be in proximity to, in order to be output.
- **SPACE *constraint2***  
 A required keyword and constraint containing non-negative, floating-point numbers in user units of length. The constraint may not be of the form “>”, “>=”, or “!=”. Specifies the dimensions of the measurement region around a *layer* polygon.
- **SQUARE**  
 Optional keyword specifying the measurement region formed by *constraint2* uses the SQUARE metric rather than the Euclidean metric.
- **CENTERS**  
 Optional keyword specifying the measurement region formed by *constraint2* is measured from the center of a given rectangle rather than from its perimeter. Distances are measured to the centers of other rectangles rather than to their edges.
- **OCTAGONAL ONLY**  
 Optional keyword that specifies the distance between the centerpoints of orthogonal rectangles is defined to satisfy the **SPACE** constraint only if the centerpoints are aligned at multiples of 45 degrees. May only be specified with the CENTERS keyword. May not be specified with the ORTHOGONAL ONLY keyword.

- ORTHOGONAL ONLY

Optional keyword that specifies the distance between the centerpoints of orthogonal rectangles is defined to satisfy the **SPACE** constraint only if the centerpoints are aligned in either the x- or y-direction. May only be specified with the CENTERS keyword. May not be specified with the OCTAGONAL ONLY keyword.

- INSIDE OF LAYER *layer3*

Optional keyword that restricts output to polygons inside of *layer3*. Refer to the algorithm description for more information. May not be specified with the CONNECTED keyword.

- [NOT] CONNECTED

Optional keyword that modifies the operation to consider only rectangles from *layer1* and *layer2*, which are on the same electrical nets (or different electrical nets if NOT is used). If this keyword is specified, the connectivity of *layer1* and *layer2*, if specified, are checked at compilation time. May not be specified with INSIDE OF LAYER.

## Description

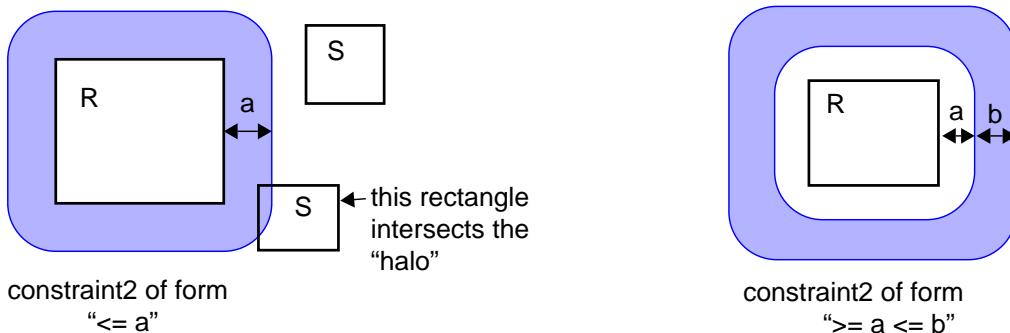
Selects orthogonal (with respect to the database axes) rectangles if they have the specified number of other orthogonal rectangles within a specified distance.

With only *layer1* specified, the operation works as follows:

1. For each orthogonal rectangle, R, on the input layer, let C be the number of other orthogonal (with respect to the database axes) rectangles having a distance to R satisfying **constraint2**.
2. If C satisfies **constraint1**, then R is output.

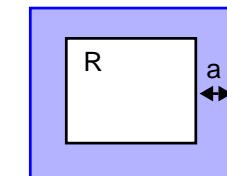
The distance between orthogonal rectangles R and S is determined using each edge of R and S in exactly the same manner as the **EXTernal** operation. The Euclidean measurement metric is used. Measurement regions having a radius or radii determined by **constraint2** are constructed around each edge of R, creating a “halo” around R. The distance between R and S is said to satisfy the **constraint2** if S intersects the halo. If R and S intersect at a singularity, then the distance is defined to be zero. See [Figure 4-344](#):

**Figure 4-344. Halo Region Using Euclidean Metric**



If **SQUARE** is specified, then the measurement region halo is constructed using the **SQUARE** metric rather than the Euclidean metric. See [Figure 4-345](#):

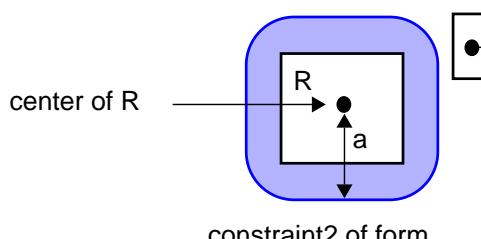
**Figure 4-345. Halo Region Using SQUARE Metric**



constraint2 of form  
“ $\leq a$ ”

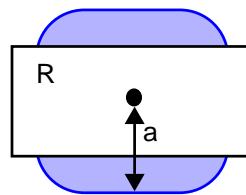
If **CENTERS** is specified, then the measurement region halo is created about the center point of R. The distance between R and S satisfies **constraint2** if the center point of S intersects the halo. See [Figure 4-346](#).

**Figure 4-346. Measurement Region Using CENTERS**



constraint2 of form  
“ $\leq a$ ”

The center of S must be within the halo region to satisfy the constraint.



When using **CENTERS**, it is possible the measurement region does not surround R, depending on the size of the constraint.

The **ORTHOGONAL ONLY** and **OCTAGONAL ONLY** keywords are used primarily for advanced pitch analysis and via alignment checks. These options are used with the **CENTERS** keyword and control how the alignment of centers is checked. **ORTHOGONAL ONLY** specifies alignment checking in the x- and y- directions. **OCTAGONAL ONLY** also does this, but also checks alignment diagonally at multiples of 45 degrees.

All **With Neighbor** and **Not With Neighbor** operations with the same **layer1**, **constraint2**, and **SQUARE** and **CENTERS** keyword specifications are executed concurrently.

If **layer2** is specified, then the algorithm changes as follows:

1. For each orthogonal rectangle R on **layer1**, let C be the number of orthogonal rectangles on **layer2** whose distance to R satisfies **constraint2**.
2. If C satisfies **constraint1**, then R is output.

## With Neighbor

The distance between two rectangles from *layer1* and *layer2* is defined to be 0 if the rectangles touch at any point, except when CENTERS is specified. If CENTERS is specified, then the distance between rectangles on *layer1* and *layer2* is measured between their centers, regardless of any overlap.

If INSIDE OF LAYER *layer3* is specified, then the algorithm is as follows:

For each polygon P on *layer3*,

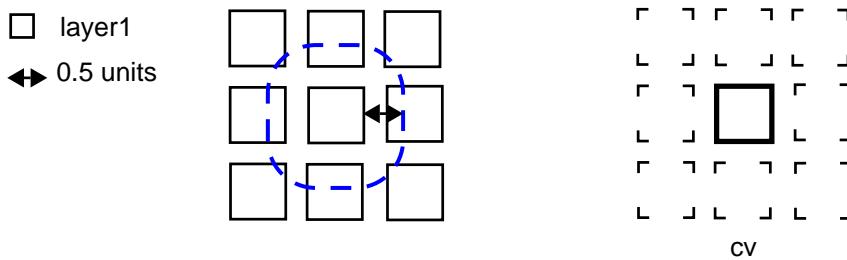
1. For each orthogonal rectangle R on *layer1*, which is inside of P,
  - o If only *layer1* is specified, let C be the number of other orthogonal rectangles on *layer1*, which are also inside of P and whose distance to R satisfies *constraint2*, or
  - o If *layer2* is specified, let C be the number of orthogonal rectangles on *layer2*, which are also inside of P and whose distance to R satisfies *constraint2*.
2. If C satisfies *constraint1*, then R is output.

If [NOT] CONNECTED is specified, then the algorithm is as follows:

1. For each orthogonal rectangle R on *layer1*
  - o If only *layer1* is specified, let C be the number of other orthogonal rectangles on *layer1* such that the rectangles are on the same (or different, if NOT is used) electrical net as R, and whose distance to R satisfies *constraint2*, or
  - o If *layer2* is specified, let C be the number of orthogonal rectangles on *layer2* such that the rectangles are on the same (or different, if NOT is used) electrical net as R, and whose distance to R satisfies *constraint2*.
2. If C satisfies *constraint1*, then R is output.

## Examples

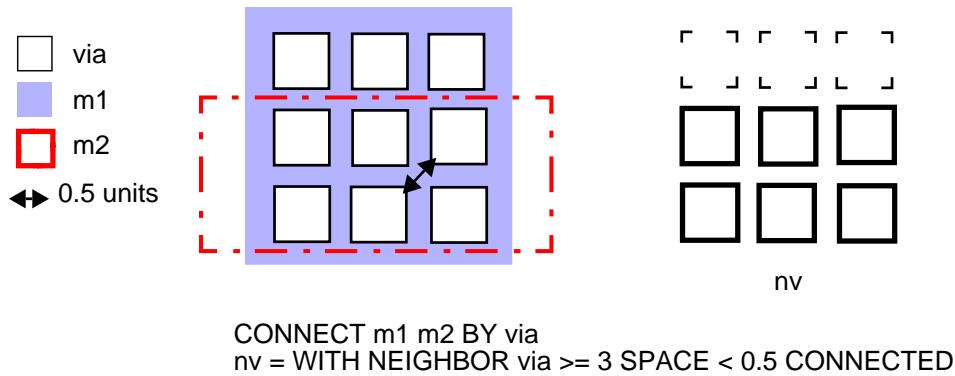
### Example 1



cv = WITH NEIGHBOR layer1 == 8 SPACE < 0.5

**Example 2**

This example demonstrates the use of the CONNECTED keyword. Only the vias on the same node are considered for the With Neighbor operation. All six of those vias satisfy the constraints given.



## With Text

Layer operation

**WITH TEXT** *layer name* [*text\_layer*] [PRIMARY ONLY] [CASE SENSITIVE]

### Parameters

- ***layer***  
A required original layer or layer set, or a derived polygon layer.
- ***name***  
A required name of a free-standing text object, which can contain one or more question mark (?) characters. The ? is a wildcard character that matches zero or more characters. The ***name*** can be a string variable (see [Variable](#)).
- ***text\_layer***  
An optional original layer or layer set where text objects are found.
- **PRIMARY ONLY**  
An optional keyword that specifies only top-cell text is to be used in the operation.
- **CASE SENSITIVE**  
An optional keyword that specifies ***name*** parameters must match by case.

### Description

Selects all ***layer*** polygons that intersect the positions of text objects having the specified ***name***. The specified parameter order must be observed to avoid ambiguity.

If *text\_layer* is specified, only text objects with the specified ***name*** on *text\_layer* are considered. The ***name*** parameter is case-insensitive by default.

[Layout Text](#) and [Layout Text File](#) text objects participate in Not With Text operations.

The [Text](#), [Text Depth](#), and [Text Layer](#) statements have no effect on With Text operations.

See also [Not With Text](#).

### Examples

```
// Select all pads larger than 5000 square microns that have  
// test-pads as a text label on layer 14:  
large_pads = pads AREA > 5000  
test_pads = large_pads WITH TEXT "test-pads" 14
```

## With Width

Layer operation

### **WITH WIDTH *layer constraint***

#### Parameters

- ***layer***  
An original layer, layer set, or a derived polygon layer.
- ***constraint***  
A required constraint interpreted as width in user units. It is one of the constraints listed in the “Constraint Notation” column of [Table 2-2](#) on page 45.

#### Description

Selects all regions of ***layer*** polygons that satisfy the width ***constraint***. This operation does not select entire input polygons unless an entire polygon meets the ***constraint***. Behaves as a non-node-preserving layer constructor.

The concept of width is somewhat difficult to define consistently. This operation provides a consistent width definition and replaces complex SVRF derivations used in determining width. In the case of angled geometry, the output conforms to intuitive notions of width. This output may be contrary (and more useful) to what you would see using **Size**, especially for layers that have received OPC corrections.

The algorithm uses the following heuristics. Consider the operation:

**metal WITH WIDTH > w**

This operation is performed on a polygon-by-polygon basis, where each polygon is divided into three possible regions—orthogonal, 45-degree, and skew, each with its own width-measurement algorithm. The **w** parameter is a numeric width. The result is the Boolean OR of the three regions. Regions are cut only along lines that are orthogonal to the database axes.

Throughout this section, the word orthogonal means orthogonal to the database axes, not orthogonal to another edge or polygon.

With Width operations using the same layer are performed concurrently when the layer data is orthogonal.

**Orthogonal regions** — For orthogonal edges, the algorithm is based upon:

**SIZE metal BY v UNDEROVER**

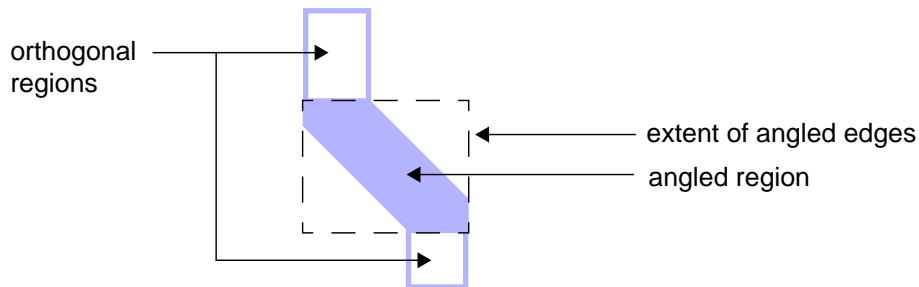
If the With Width constraint is  $> w$ , then  $v = w/2$ . If the With Width constraint is  $\geq w$ , then  $v = w/2 - \text{delta}$ , where delta is a small number in comparison to  $w$ . All other constraints are derived from these definitions. For example, the set of polygons selected by a With Width  $\leq w$  operation is the logical complement of the set selected by  $> w$ .

**45-degree regions** — For 45-degree edges, an *angled* region is determined based on the extent of the 45-degree edges. The angled region is evaluated separately from the original polygon, leaving completely orthogonal polygons (possibly empty) that are handled by the orthogonal

region algorithm described previously. Note that this algorithm produces identical results for the orthogonal regions as the Size UNDEROVER operation does, so any orthogonal region that is part of the angled region is handled consistently.

The angled region, plus a sufficient orthogonal region around it, are handled by the angled algorithm. See [Figure 4-347](#).

**Figure 4-347. Angled Region**



The angled algorithm is based upon iterative measurements like this:

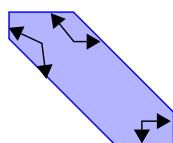
**INT layer < x OPPOSITE REGION**

which are subtracted from the angled region. The portions that are subtracted from the angled region depend upon the specified **constraint** in the With Width operation. The iteration is based upon the angle of separation between various pairs of edges in the angled region. The iteration stops when there is either no more measurement region output to subtract, or the angled region is empty.

The INT *layer < x OPPOSITE REGION* measurement is not identical to that of the usual Internal check with the same keywords. The input has only orthogonal and 45-degree edges; therefore edge pairs that are appropriate for internal measurement have angles of 135, 90, 45, or 0 degrees between them. Measurements are taken as follows:

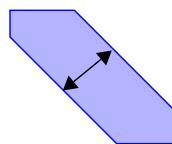
- Edges having an angle 135 or 90 degrees with respect to each other are not measured and are disregarded.

edges separated  
by 135 or 90  
degrees are not  
measured



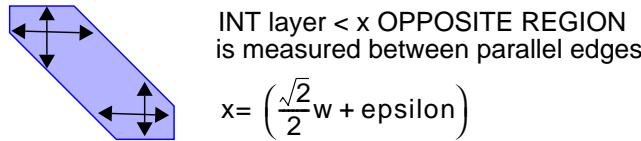
(these figures show a  
subset of all possible  
edge combinations in  
the respective categories)

- Edges having an angle of 0 degrees with respect to each other (parallel edges) are measured at  $x = w + \text{epsilon}$  (where  $w$  is the width value, and epsilon is a small number compared to  $w$ ) and produce the same output as a customary Internal check with the OPPOSITE REGION keywords.



INT layer < x OPPOSITE REGION  
is measured between parallel edges  
 $x = w + \text{epsilon}$

- Edges having an angle of 45 degrees with respect to each other have  $x = ((\sqrt{2}/2) * w) + \text{epsilon}$ .



The measurement and region is formed as follows. Assume one edge is orthogonal (horizontal or vertical) and the other is at a 45-degree angle with the x-axis.

- If the OPPOSITE measurement region from the orthogonal edge intersects the 45-degree edge, or vice-versa, this produces an output region.
- Or, if the OPPOSITE measurement region from the 45-degree edge intersects the orthogonal edge, this forms the output region.
- Otherwise, no output is produced.

See [Example 3](#).

**Skew regions**—For polygons that contain non-orthogonal, non-45 degree edges, a *skew* region is determined based on the extent of the skew edges. The skew region is cut from the original polygon, leaving angled (orthogonal and 45-degree edges) polygons (possibly empty) that are handled by the angled algorithm. This is a similar process as described previously for angled regions. The skew region, plus a sufficient angled region around it, are handled by the operation:

#### **SIZE *layer* BY *v* UNDEROVER TRUNCATE *v***

where *v* is determined as previously shown.

The operation performs a Boolean AND of the input *layer* with the result geometry prior to output in areas near skew edges on the input *layer*. This step increases the accuracy of the output.

See also [Not With Width](#) and [Internal](#).

## Examples

### Example 1

Basic layer derivation using With Width:

```
// derive polysilicon regions having width <= .10
narrow_poly = poly with width <= .10

// derive input polysilicon polygons having width <= 10
narrow_poly = poly with width <= .10
orig_narrow_poly = narrow_poly interact poly
```

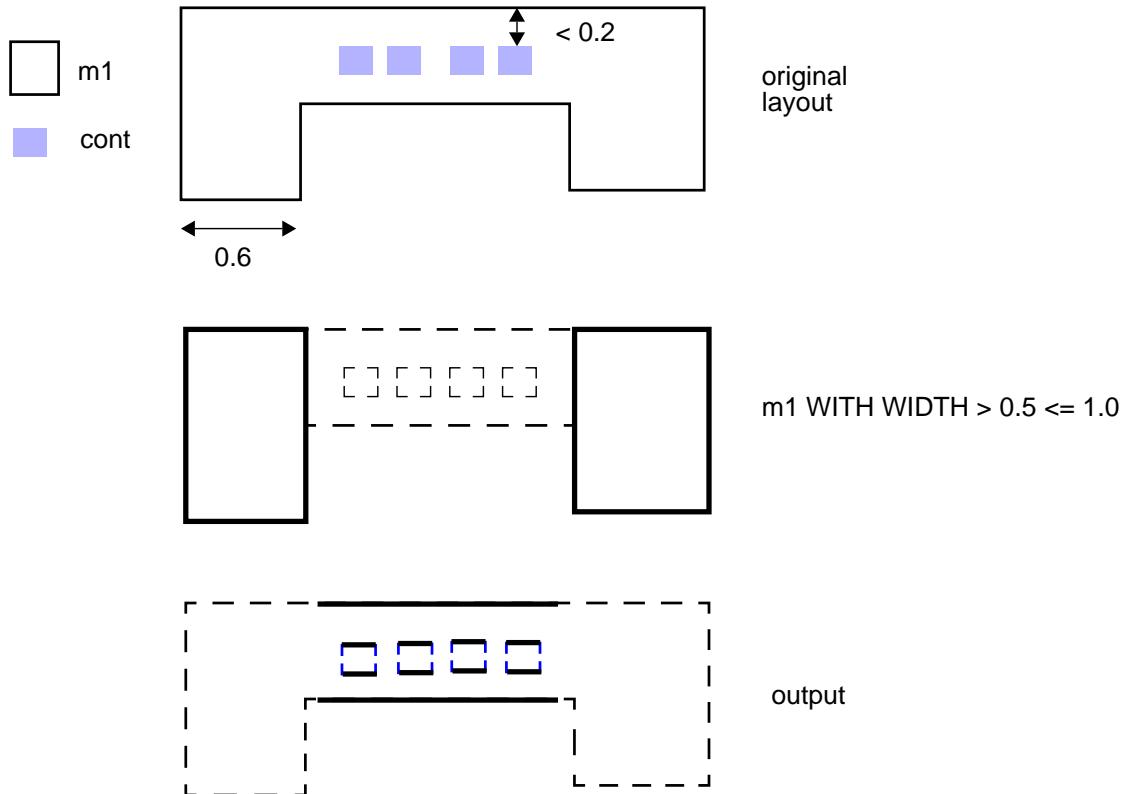
## With Width

---

### Example 2

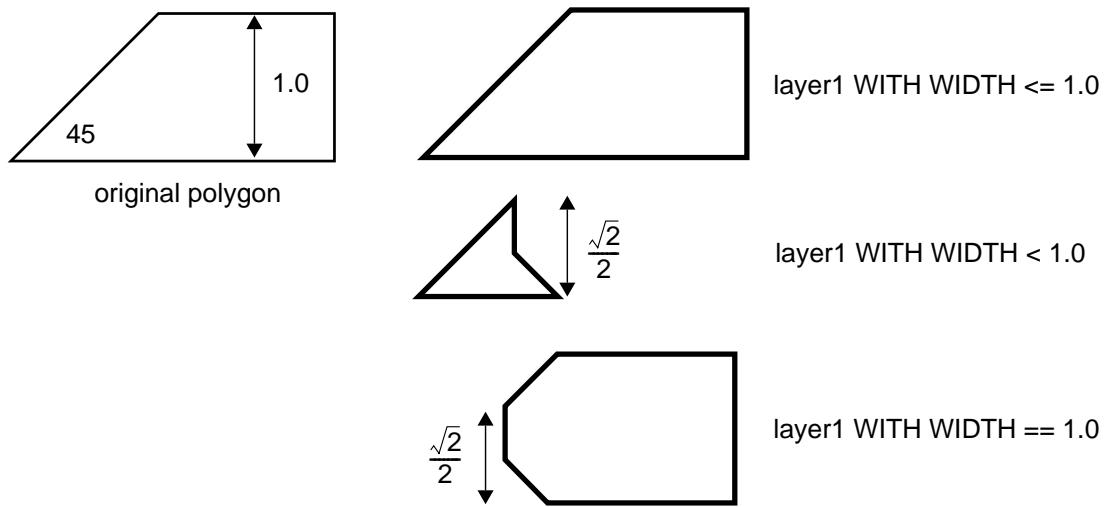
Assume your design rule states that the minimum enclosure of contact by metal1 of width greater than 0.5, and less than or equal to 1.0, is 0.2. You could write the rule this way:

```
wide_m1_over_cont {  
    wide_m1 = m1 INTERACT (m1 WITH WIDTH > 0.5 <= 1.0)  
    ENC cont wide_m1 < 0.2 ABUT < 90 SINGULAR  
}
```



**Example 3**

This example shows how With Width treats angled edges for various width constraints.



## XOR

Layer operation

**XOR *layer1***

**XOR *layer2 layer3***

### Parameters

- ***layer1***  
A required original layer or layer set.
- ***layer2***  
A required original layer or layer set or a derived polygon layer.
- ***layer3***  
A required original layer or layer set or a derived polygon layer.

### Description

Selects all polygon areas common to exactly one polygon. The one-layer operation is equivalent to the [AND](#) *layer1 == 1* operation. The two-layer operation generates output equivalent to the polygon data on *layer3* if *layer2* is empty. If *layer3* is empty, the two-layer operation generates output equivalent to the polygon data on *layer2*.

The one-layer Boolean XOR operation does not merge overlapping polygons on the original input layers.

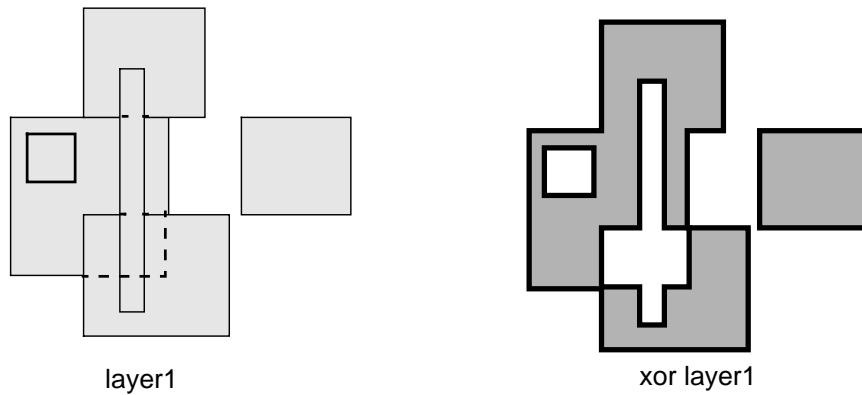
The single-layer XOR operation can give unexpected results in hierarchical operation for databases that have duplicate placements with overlapping geometry. Calibre selectively discards duplicate placements in many cases as part of its internal optimization. Databases with such characteristics are better handled using Pyxis Layout.

See also [NOT](#) and [Resolution](#).

### Examples

#### Example 1

The one-layer Boolean XOR operation shown in Figure 4-348 selects all *layer1* polygon areas common to exactly one *layer1* polygon.

**Figure 4-348. One-Layer Boolean XOR****Example 2**

The two-layer Boolean XOR operation shown in Figure 4-349 selects all layer2 and layer3 polygon areas common to exactly one layer2 or layer3 polygon.

Reversing the order of layer2 and layer3 in the Boolean XOR operation shown in Figure 4-349 does not produce any different output because the Boolean XOR is commutative. For example,

**xor layer3 layer2**

produces the same output as shown in Figure 4-349.

**Figure 4-349. Two-Layer Boolean XOR**



# Chapter 5

## Built-In Languages

---

### Common Features of SVRF Built-In Languages

SVRF supports several built-in languages for user-defined calculations. These languages are employed within various SVRF statements including the [Device](#) statement, the [Device Layer](#) operation, the LVS Reduce family of statements, the [LVS Property Initialize](#) statement, and the [Trace Property](#) statement. They are used for tasks such as property computations, device reduction computations, and device property tracing.

These are the built-in languages used in SVRF:

- [Device Property Computation Built-In Language](#)
- [Device Annotation Built-In Language](#)
- [Device Reduction Effective Property Computation Language](#)
- [LVS Property Initialize Built-In Language](#)
- [Trace Property Computation and Reporting Language](#)

### General Characteristics

The general characteristics for all of the built-in languages are described in Table 5-1 and the complete set of numeric functions used in the languages is described in Table 5-2. The individual languages and their unique features are described later in this chapter.

**Table 5-1. Built-In Language General Characteristics**

Characteristic	Description
Language Style	The built-in languages are designed as a blend of the expression and computational statement style of C, together with conventions common to the rest of the SVRF rule file language. Added to this are necessary declarations and functions.
Structure	The languages are structured as a sequence of statements. The allowed statements and their prescribed order depends on the language you are using. Assignment statements may appear, as well as conditional IF ... ELSE statements. There are no loop statements in the languages; however, the TVF interface to the Device operation supports Tcl loops.

**Table 5-1. Built-In Language General Characteristics (cont.)**

<b>Characteristic</b>	<b>Description</b>
Statement Placement and Continuation	In keeping with the previously-established syntactic conventions of the SVRF rule file language, semicolons or other separating devices are not used between statements. The language elements are recognized by their keywords. For some statements such as PROProperty, PROProperty STRING, EFFECTIVE, EFFECTIVE STRING, WARN, DEBUG, IF, and ELSE, the keyword must be the first word on a line. Assignment statements are recognized by the “=” operator, which is always the second item in the statement. Statements may be continued onto multiple lines by breaking them at any whitespace. Continuation characters are not employed. Common practice dictates that each statement begins on a new line, but this is not generally required.
Reserved Keywords	The keywords of the language such as PROProperty, PROProperty STRING, EFFECTIVE, EFFECTIVE STRING, WARN, DEBUG, IF, and ELSE are reserved. In keeping with general rule file syntax, if a word is surrounded by single (‘) or double (“) quotes, it is not taken to be a keyword, but rather a normal identifier for a property name, local (temporary) variable, or possibly a string-valued argument to a function, for functions that support this. Thus, the reserved nature of keywords places no restrictions on property or local variable naming.
Optional Keyword and Function Spellings (Keyword Abbreviations)	The longer keywords and function names often have a shortened form consisting of a set of letters from the complete name. In this document, the letters in the shortened form are shown in uppercase, with the remaining letters in lowercase. PROProperty indicates that the spelling is either PROPERTY or PROP. If any of the optional letters are used, they must all be used. Thus, PROPERT is not a valid shortening of PROProperty.
Case Sensitivity	The language is case-insensitive except for Device TVF elements. All uppercase letters are converted to lowercase for internal purposes. For example, a variable may be referred to as “maxL”, “MaxL”, “MAXL” (or any similar combination) interchangeably within the same procedure. The use of mixed upper- and lowercase letters in this document is used only to show allowed abbreviations. The <a href="#">Layout Preserve Case</a> statement supports the preservation of declared property case.
Comments	The comment conventions are the same as for the rest of the rule file. Any text beginning with a double slash (//) through the end of the line is ignored. Any text inside a pair of /* and */ markers is also ignored. Rule check comments (@) are not used.
Commas	All commas shown in the language elements are required.

**Table 5-1. Built-In Language General Characteristics (cont.)**

Characteristic	Description																
Flow Control	<p>The only flow control is provided by the IF and IF ELSE statements, which have the same meaning and use as in C. These statements have the following forms:</p> <p style="padding-left: 40px;"><i>IF (logical-expression) statement</i>  <i>IF (logical-expression) statement ELSE statement</i></p> <p>There are no looping statements in the languages.</p>																
Data Types	<p>The built-in languages handle numeric and character-string properties (the LVS Property Initialize language only handles numeric values, which is described under “<a href="#">LVS Property Initialize Built-In Language</a>” on page 1867). All numeric literal values are specified in double-precision floating-point. Calculations specified by the user procedure are also carried out in double-precision. Discrepancy results, if any, are returned to the LVS comparison phase in single-precision floating-point.</p>																
Constants	<p>Numeric constants are constructed according to the same rules as in the rest of the rule file, namely as valid C integer, float, or double constants. Here are some examples:</p> <p style="text-align: center;">3 3.0 -2.5 4.6e-10 5e8 5E9 0 1</p>																
Local Variables (Temporaries)	<p>Intermediate numeric values may be assigned to local variables. For compatibility with rule file style, it is neither necessary nor possible to declare such variables before use. Any name may be used that does not conflict with property names or process variables. However, it is best to avoid names beginning with “temp” or “init” since these names are used in DEBUG monitoring output to identify temporary variables. Although the compiler and interpreter are not confused by your use of these names, such names may confuse human readers. Note that assignment of string values to local variables is only supported in the Device property computation built-in language.</p>																
Operators	<p>A subset of the operators of C are available. They have the same meaning and precedence as in C, and are listed below in order of decreasing precedence.</p> <table style="margin-left: 40px;"> <tbody> <tr> <td>( )</td> <td>grouping</td> </tr> <tr> <td>+ - !</td> <td>unary plus, unary minus, logical negation</td> </tr> <tr> <td>* /</td> <td>multiplication, division</td> </tr> <tr> <td>+ -</td> <td>addition, subtraction</td> </tr> <tr> <td>&lt; &lt;= == &gt; = &gt; !=</td> <td>relational operators: lt, le, eq, ge, gt, ne</td> </tr> <tr> <td>&amp;&amp;</td> <td>logical and</td> </tr> <tr> <td>  </td> <td>logical or</td> </tr> <tr> <td>=</td> <td>assignment</td> </tr> </tbody> </table>	( )	grouping	+ - !	unary plus, unary minus, logical negation	* /	multiplication, division	+ -	addition, subtraction	< <= == > = > !=	relational operators: lt, le, eq, ge, gt, ne	&&	logical and		logical or	=	assignment
( )	grouping																
+ - !	unary plus, unary minus, logical negation																
* /	multiplication, division																
+ -	addition, subtraction																
< <= == > = > !=	relational operators: lt, le, eq, ge, gt, ne																
&&	logical and																
	logical or																
=	assignment																

**Table 5-1. Built-In Language General Characteristics (cont.)**

<b>Characteristic</b>	<b>Description</b>
Parentheses	Parentheses can be used in expressions to override normal operator precedence.
Numeric Expressions	A numeric expression is a combination of numeric constants, numeric variables, and the following operators in order of precedence from highest to lowest:  ( )        grouping + -        unary * / + -    binary
String Expressions	Only trivial string expressions are supported. That is, a string expression may only consist of a single string literal, string-valued variable, or string-valued function reference. Literal strings are enclosed in double quotes (" "), for example: "A2".
Logical Expressions	Basic logical expressions are formed by combining numeric expressions using the relational comparison operators. Logical expression may be further combined using parenthesis together with the logical and (&&) and logical or (  ) operators. Logical expressions may only appear within the parentheses following the keyword IF. String expressions may not directly participate in logical expressions.
Assignment Statements	The assignment statement has the following form:  $local\_variable\_or\_property\_name = numeric\_expression$ $property\_name = string\_expression$  The Device property computation built-in language also allows string expressions to be assigned to local variables, but the other languages do not. Only one = operator is allowed per assignment statement. Numeric-valued variables may not be reassigned as string valued variables, and string-valued variables may not be reassigned as numeric.
Statement Grouping	Braces {} may be used to group one or more statements into a single statement, as in C.

## Numeric Functions for Built-in Languages

Table 5-2 gives the numeric functions available in the built-in languages:

**Table 5-2. Numeric Functions for Built-in Languages**

Function	Description
Absolute Value	<p>The syntax is as follows:</p> $\text{ABS}(\text{numeric-expression})$ <p>Returns the absolute value of the specified numeric expression.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>ABS(-1.5) returns 1.5</li> <li>ABS(2.5) returns 2.5</li> </ul>
Exponent	<p>The syntax is as follows:</p> $\text{EXP}(\text{numeric-expression})$ <p>Returns the value of <math>e</math> (Napier's constant, the base of natural logarithms) raised to the power of the numeric expression.</p>
Logarithm	<p>The syntax is as follows:</p> $\text{LOG}(\text{numeric-expression})$ <p>Returns the natural logarithm of the specified numeric expression. The value of the numeric expression must be greater than zero.</p>
Power	<p>The syntax is as follows:</p> $\text{POW}(\text{numeric-expression}, \text{numeric-expression})$ <p>Returns the value of the first expression raised to the power of the second expression. If the first expression is zero, the second must be positive. If the first expression is negative, the second expression must be an integer.</p>
Square Root	<p>The syntax is as follows:</p> $\text{SQRT}(\text{numeric-expression})$ <p>Returns the positive square root of the specified numeric expression. The value of the numeric expression must be non-negative.</p>

**Table 5-2. Numeric Functions for Built-in Languages (cont.)**

Function	Description
Truncate	<p>The syntax is as follows:</p> $\text{TRUNC}( \text{numeric-expression} )$ <p>Returns the result of truncating the fractional part of the specified numeric expression. Truncation is toward zero. The absolute value of the numeric expression must not exceed 2,147,483,647.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>TRUNC(2.1) returns 2</li> <li>TRUNC(2.9) returns 2</li> <li>TRUNC(-2.1) returns -2</li> <li>TRUNC(-2.9) returns -2</li> <li>TRUNC(9999999999) returns Unknown, because the argument exceeds 2147483647.</li> </ul>

## Device Property Computation Built-In Language

The [Device](#) statement supports user-defined device property computations. This optional section of a Device statement consists of a program that defines device properties. It has all the features of the built-in languages described in [Table 5-1](#) and [Table 5-2](#). The specific aspects of the property computation language (sometimes referred to simply as the built-in language) are described in the following subsections.

If you provide your own property specification, you assume responsibility for calculating *all* properties of the device, even if default properties are provided by device recognition for the specific device.

Only one property specification is allowed per Device operation. It lists the names and types of the properties to be computed for that device and specifies the method of computation from available data.

Additional information regarding device property computation, including efficient programming techniques, debugging, and so forth, is found under “Property Computation” in the [Calibre Verification User’s Manual](#).

The device property computation language is often used for MOS device well proximity and shallow trench isolation (STI) stress effect calculations. The [device::enclosure\\_measurements](#) function is best used for this purpose.

You can use Tcl programming for complicated calculations when you use the [TVF Functional Interface for Device Property Calculation](#).

These topics are covered in this section:

Writing a Device Property Computation Program . . . . .	1777
Property Computation Language Characteristics . . . . .	1779
Data Sources . . . . .	1780

DEBUG Statement . . . . .	1781
PROPerty Statement . . . . .	1781
Summary of Device Property Computation Functions . . . . .	1782
Pin and Layer References . . . . .	1786
Function Reference . . . . .	1786
TVF Functional Interface for Device Property Calculation . . . . .	1827
Well Proximity Calculation Methods and Examples . . . . .	1833

## Writing a Device Property Computation Program

By default, the only properties available for a PMOS transistor are the effective width (W) and length (L) of the gate. A typical Device operation appears as follows, with the effective width parameter specified as 0.5:

```
DEVICE mp (pmos) gate gate (G) diff (S) diff (D) [ 0.5 ]
```

Suppose you want to compute not only W and L, but also the areas of the source (AS) and drain (AD) pins. To do this, you must not only add the new computations for AS and AD, but you must provide computations for W and L. It is not possible to combine the default and new methods of computation, so you must calculate all desired properties. Using the built-in language, the altered Device operation might look as follows:

```

1 device mp (pmos) gate gate (G) diff (S) diff (D)
2 [
3     property W, L, AS, AD, COLOR
4     weffect = 0.5
5     AS = area(S)
6     AD = area(D)
7     W = (perim_co (G,diff) + perim_in (G,diff)) / 2
8     L = perim_outside(G, diff) / 2
9     if (bends(gate) > 0)
10    {
11        if (W > L)
12            W = W - weffect * bends(gate) * L
13        else
14            L = L - weffect * bends(gate) * W
15    }
16    COLOR = 'red'
17 ]

```

This example illustrates many of the features of the language. The line numbers on the left are not part of the language. The language is case-insensitive. In this example, only the property and pin names are capitalized.

Line 1 — Contains all of the Device statement, except the property specification, which is contained between the matching square brackets on lines 2 and 17.

Line 3 — Contains the property statement that declares the five properties computed and gives their names. Refer to “[Structure](#)” in [Table 5-3](#) for the elements of a property computation program.

Line 4 — Assigns value 0.5 to the temporary variable weffect. This variable is later used in the adjustment of W or L on lines 12 and 14.

Line 5 — Computes the value of property AS as the area of the pin named S. The pin S is declared on line 1. The area( ) function is one of the built-in functions available for delivering summarized geometric data.

Line 6 — Computes the value of property AD as the area of the pin named D.

Line 7 — Computes the physical width of the gate as half of the perimeter of the gate pin, G, which is coincident with or inside of the perimeter of shapes on the diff layer. The functions perim\_co( ) and perim\_in( ) are abbreviated forms of perimeter\_coincide( ) and perimeter\_inside( ), respectively.

Line 8 — Computes the physical length of the gate as half of the perimeter of the gate pin, G, which is outside of the perimeter of shapes on the diff layer.

Line 9 — Tests whether there are any bends in the gate pin. If there are, then a further test on line 11 determines whether the gate pin is wider than it is long. If wider, an adjustment to the effective width, W, is made on line 12, otherwise an adjustment to the effective length, L, is made on line 14.

Line 16 — Assigns the literal string value red to property COLOR.

In general, you should only extract properties that you need for the purpose of your current run. Many properties are used in post-layout simulation but are not needed for LVS debugging. You should avoid extracting such properties when you do not need them. Rule file pre-processing directives (conditionals, like #IFDEF) are useful in controlling the properties that you extract.

You should also use *local* property measurements. These are measurements that use a seed layer rather than a pin layer.

## Property Computation Language Characteristics

The details of the Device statement built-in language are described in the following sections. Additional examples are provided. The following table shows information specific to the built-in language, which is in addition to the information provided in [Table 5-1](#) and [Table 5-2](#).

**Table 5-3. Property Computation Built-In Language Specifics**

Element	Description
Structure	<p>The built-in language has a required sequence of statements, shown here:</p> <pre>[   DEBUG Statement    // optional, must be first if present   PROPerty Statement // required   property computations // the properties in the                         // PROPerty statement must be                         // defined in the program ]</pre> <p>There is no WARN statement in this language. Refer to “<a href="#">Structure</a>” in <a href="#">Table 5-1</a> for general notes on the structure of the language.</p>
Macros	Property computations can be written as macros that can be called multiple times in a rule file. See the “ <a href="#">Macros</a> ” section for details.
Local Variables	You can assign intermediate values to local variables within a property specification. Numeric, string, and complex type local variables are supported. Types for these variables are determined by the functions that are used. Once a type is associated with a variable, the type may not change.
Assignment Statements	<p>Assignment statements have the following forms:</p> <pre>local_variable_or_property_name = numeric_expression local_variable_or_property_name = string_expression local_variable_or_property_name = function</pre> <p>The last case is for functions that return complex-type variables. You can specify only one = operator per assignment statement.</p> <p>Note: Variables types cannot be changed after the initial assignment. Examples:</p> <pre>A = 5                                //numeric B = AREA(seed) + A + 7                 //numeric C = "ABC"                             //string D = TEXT_STRING( txtlay1, "default1" ) //string E = DFM_NUMERIC_ARRAY( sd_prop, "NC" ) //array</pre>
Units of measurement	For information about which units to use for representing physical quantities such as length and area, refer to “ <a href="#">Units of Measurement</a> ” in the <a href="#">Calibre Verification User’s Manual</a> .

**Table 5-3. Property Computation Built-In Language Specifics (cont.)**

Element	Description
TVF interface	The <a href="#">TVF Functional Interface for Device Property Calculation</a> is governed by the rules of Tcl code, which are somewhat different from the Device property computation built-in language. When working with TVF Function blocks, ensure that you use valid Tcl code.
Pre-processor directives	Pre-processor directives, or conditionals, are permitted in Device property calculations. These can help you to control which properties get calculated for the type of run you are using. See <a href="#">“Pre-Processor Directives”</a> on page 56.

## Data Sources

Data used in the computation can come from any of these sources:

- Constants — You can use numeric, floating-point constants, and string literals directly within the property specification.
- Numeric Process Variables — These values are accessed by using a rule file [Variable](#) or Pyxis process variable just as any other variable within the property specification. Pyxis Process variables must exist when the rule file is loaded.
- String Process Variables — Rule file and Pyxis Process string variables may be assigned to property variables (names declared in the PROPerty statement) or assigned to local variables. No other use of string rule file or string process variables is supported.

Example:

```
VARIABLE procVar "characters"
/* define a rule file variable with string value */

DEVICE D seed pin(POS) pin(NEG) [
    PROPERTY str
    str = procVar
    // property str is assigned string value "characters"
    localStr = procVar
    // local variable is assigned string value "characters"
    procVar = "string"
    // COMPILER ERROR, cannot assign a string to a process variable
]
```

As indicated in the example, rule file (and Pyxis Process) variables may never appear as the left-hand side of an assignment. Doing so results in a compiler error.

- Instance Data — Geometric, connectivity, and text data associated with an instance are accessed by the provided built-in functions. Examples are the AREA and PERIMETER functions. Properties generated on specific shapes using a [DFM Property](#) operation are also available to commands that support them. A complete list of the functions and their definitions is given under “[Summary of Device Property Computation Functions](#)” on page 1782.

## **DEBUG Statement**

Tracing of the property computation can be useful in finding errors during the development of new property computation code. Tracing is controlled by the DEBUG statement, which is optional. If present, it must be the first statement in the program. It has the following form:

**DEBUG range1 [ , range2 , ... rangeK ]**

where each range is either a device instance number, or a pair of device instance numbers separated by a hyphen (-). At least one range must be specified, although it can contain only a single instance. For example, the statement:

```
DEBUG 0-2, 50
```

causes a step-by-step report of the property computation to be printed for each of the device instances 0, 1, 2, and 50. The trace can produce much output, so the use of small ranges is recommended. You can obtain the instance numbers from LVS discrepancy reports in nmLVS-H.

The INSTance() built-in function discussed under “[Summary of Device Property Computation Functions](#)” provides a method of determining the instance number of a particular device, which is often helpful in debugging. A complete discussion of using the DEBUG statement with examples is given under “[Debugging Property Computations](#)” in the *Calibre Verification User’s Manual*.

## **PROPerty Statement**

The PROPerty statement declares the names of the properties to be computed. This statement is required and, in the absence of the DEBUG statement, must be the first statement in a property computation. It has the following form:

**PROPerty prop\_name\_1, prop\_name\_2, ...**

In the following example, four properties are declared with names W, L, SA, and DA:

```
PROPERTY W, L, SA, DA
```

Property names are treated as variables within computational statements. The final value assigned to the property name is the value of the property associated with the instance.

Property names must begin with a letter character followed any legal SPICE characters. See “[General SPICE Syntax Summary](#)” in the *Calibre Verification User’s Manual* for more details. The rule file compiler will allow you to specify property names that are not SPICE compliant,

so care should be taken in choosing names. The Calibre nmLVS SPICE reader will complain about illegal property names that appear in a netlist generated from bad property names in the rule file.

## Summary of Device Property Computation Functions

A variety of built-in functions provide access to geometric and connectivity information about an instance. You can use these functions within numeric, string, or complex expressions. Numeric results can be used as an argument to a relational operator.

The section “[Pin and Layer References](#)” on page 1786 describes how references to pin names and layers are used in the device property functions.

Table 5-4 lists the data retrieval functions available for the built-in language, along with a description of their use.

**Table 5-4. Device Property Computation Built-In Functions**

Function	Description
<b>Area functions</b>	These functions do area calculations.
<a href="#">area</a>	Returns the total area of shapes that are part of the specified pin, or on the specified layer.
<a href="#">area_common</a>	Returns the total area that is common to shapes on both of the pins or layers referenced.
<b>Counting functions</b>	These functions do counts of various quantities.
<a href="#">bends</a>	Returns the total bends in the shapes of the specified pin or on the specified layer. The result is expressed in units of right angles.
<a href="#">count</a>	Returns the total number of shapes on the specified layer or in the specified pin.
<b>Enclosure functions</b>	These are specialized functions for calculating device enclosures. These functions are useful for well proximity and stress effect calculations. Refer to “ <a href="#">Well Proximity Calculation Methods and Examples</a> ” on page 1833 for examples.
<a href="#">device::enclosure_measurements</a>	Returns a vector of numbers that correspond to edge enclosure measurements. This function must be used in a TVF Function block, and has performance advantages over the other enclosure implementations.
<a href="#">enclosure_parallel</a>	Returns a vector of numbers that correspond to trapezoidal dimensions of average enclosure distances and widths. The device::enclosure_measurements function gives much better performance for well proximity measurements and is recommended for that application.

**Table 5-4. Device Property Computation Built-In Functions (cont.)**

Function	Description
<a href="#">enclosure_parallel_multifinger</a>	Returns a vector of numbers that correspond to trapezoidal dimensions of average enclosure distances and widths for multifinger devices. The device::enclosure_measurements function gives much better performance for well proximity measurements and is recommended for that application.
<a href="#">enclosure_perpendicular</a>	Returns a vector of numbers that correspond to trapezoidal dimensions of average enclosure distances and widths. The device::enclosure_measurements function gives much better performance for well proximity measurements and is recommended for that application.
<a href="#">enclosure_perpendicular_multifinger</a>	Returns a vector of numbers that correspond to trapezoidal dimensions of average enclosure distances and widths for multifinger devices. The device::enclosure_measurements function gives much better performance for well proximity measurements and is recommended for that application.
<a href="#">enclosure_vector</a>	Returns a vector of numbers that correspond to trapezoidal dimensions for polygons. This function has performance disadvantages compared to the other enclosure functions, but is useful in simple cases. The device::enclosure_measurements function gives much better performance for well proximity measurements and is recommended for that application.
<a href="#">device::eval_dfm_func</a>	Enables access to DFM Function TABLE and MATRIX functions. This function is only used within a TVF Function block.
<b>DFM Property Data Retrieval Functions</b>	These functions are used in conjunction with the DFM Property operation to retrieve property values for use within Device property computations. Refer to “ <a href="#">DFM Property Data Retrieval Functions</a> ” on page 1788.
<a href="#">dfm_numeric_value</a>	Provides access to a DFM property from a single polygon on a pin, pin layer, or auxiliary layer. The DFM property is accessible as a regular numeric value.
<a href="#">dfm_numeric_array</a>	Provides access to a DFM property from a set of polygons on a pin, pin layer, or auxiliary layer. The set of properties is accessible through a TVF object created by this function.
<a href="#">dfm_vector_value</a>	Provides access to a DFM vector property from a single polygon on a pin, pin layer, or auxiliary layer. The set of properties is accessible through a TVF object created by this function.

**Table 5-4. Device Property Computation Built-In Functions (cont.)**

<b>Function</b>	<b>Description</b>
<code>dfm_vector_array</code>	Provides access to a DFM vector property from a set of polygons on a pin, pin layer, or auxiliary layer. The set of properties is accessible through a TVF object created by this function.
<code>device::dfm_vec_measurements</code>	Converts DFM-style rectilinear measurement vectors into Device enclosure-vector-style width and length measurements. This function is only used within a TVF Function block.
<code>device::scaled_dfm_vec_measurements</code>	Converts DFM-style rectilinear measurement vectors into Device enclosure-vector-style width and length measurements, and scales the results. This function is only used within a TVF Function block.
<b>Perimeter functions</b>	These functions do perimeter calculations. Refer to “ <a href="#">Perimeter Functions</a> ” on page 1818.
<code>perimeter</code>	Returns the total length of the perimeter of the shapes in the specified pin or on the specified layer.
<code>perimeter_inside</code>	Returns the total length of the parts of perimeters on the first pin or layer that lie strictly inside shapes of the second pin or layer.
<code>perimeter_outside</code>	Returns the total length of the parts of perimeters on the first pin or layer that lie strictly outside shapes of the second pin or layer.
<code>perimeter_coincide</code>	Returns the total length of the parts of perimeters on the first pin or layer which coincide with the perimeter of the second pin or layer.
<code>perimeter_coincide_inside</code>	Returns the total length of the parts of perimeters on the first pin or layer that coincide with parts of the perimeters of shapes of the second pin or layer and where the shape of the first layer is inside the shape of the second layer.
<code>perimeter_coincide_outside</code>	Returns the total length of the parts of perimeters on the first pin or layer that coincide with parts of the perimeters of shapes of the second pin or layer and where the shape of the first layer is outside the shape of the second layer.
<b>Instance function</b>	
<code>instance</code>	Returns the instance number of the current device instance. This function allows conversion of these numbers into a property value for use in other contexts.
<b>Net functions</b>	These functions are useful for detecting pins that are connected to special nets such as power or ground.
<code>named_net</code>	Returns the number of a named net. This is not implemented hierarchically.

**Table 5-4. Device Property Computation Built-In Functions (cont.)**

Function	Description
<code>pin_net</code>	Returns the net number of a specified pin. In hierarchical applications, this function returns node number local to the cell, which is a limitation.
<b>Unit functions</b>	These return units of measure.
<code>precision</code>	Returns the current value of the process precision (1000 by default).
<code>unit_capacitance</code>	Returns the current value of the unit capacitance as specified in the rule file.
<code>unit_length</code>	Returns the current value of the process <a href="#">Unit Length</a> , 1E-6 by default.
<code>unit_resistance</code>	Returns the current value of the unit resistance as specified in the rule file.
<b>Location function</b>	
<code>x_location</code>	Returns the x coordinate in user units associated with the given pin or layer.
<code>y_location</code>	Returns the y coordinate in user units associated with the given pin or layer.
<b>Vector function</b>	
<code>sum</code>	Returns the sum of vector elements. This function manipulates output from operations that produce lists of values.
<b>Text functions</b>	
<code>text_string</code>	Returns string values of text objects in the layout that intersect the device seed shape.
<code>text_numeric</code>	Returns numeric values of text objects in the layout that intersect the device seed shape.
<b>Device TVF functions</b>	The Tcl Verification Format interface for Device property calculations is described under “ <a href="#">TVF Functional Interface for Device Property Calculation</a> ” on page 1827.
<code>TVF_numeric_function</code>	Provides access to a numeric value returned by a Tcl procedure defined within a TVF Function definition.
<code>TVF_string_function</code>	Provides access to a string value returned by a Tcl procedure defined within a TVF Function definition.

## Pin and Layer References

Most of the functions take one or more arguments that are references to either a layer or a pin. When referencing a pin, the compiler uses the name specified in the Device operation. When referencing a layer, the compiler uses the name or number of the layer only if it matches the name or number in the layer list of the Device operation. If the reference could be interpreted as either a pin or a layer, the pin interpretation is chosen.

When referencing a layer, you do not reference all shapes on the layer, but only those shapes of the layer that touch or overlap the seed shape of the current device instance. For example, the statement:

```
AREA(lay1)
```

returns the total area of all shapes on layer lay1 that touch or overlap the current instance's seed shape.

When a pin name is used, the function returns summary information about all shapes that are part of that pin. Because auxiliary layers have no pin names, you can reference them only by layer name or number.

The shapes on layers are merged prior to device extraction. For example, if a pin had been formed by two overlapping shapes drawn on original layer lay5, the perimeter returned by PERIMETER(lay5) would be the perimeter of the merged shape, not the sum of the perimeters of the two original overlapping shapes. Similarly, functions that access [DFM Property](#) operation values return properties from individual shapes.

Local property measurements are function calls that involve the seed layer and, optionally, one pin layer or auxiliary layer. Non-local property measurements in Device operations are function calls that involve a pair of pin layers (or pins), a pin layer and an auxiliary layer, or two auxiliary layers.

Non-local measurements tend to degrade the hierarchy and should be avoided if the same task can be achieved with local measurements. For example, the following is a non-local measurement:

```
DEVICE MP pgate poly(G) psd(S) psd(D) nw(B) <aux1>
[
  ...
  z = perimeter_coincide( G, aux1 )
  ...
]
```

The following is a local measurement that may be equivalent and preferable:

```
z = perimeter_coincide( pgate, aux1 )
```

## Function Reference

The section provides the descriptions for all of the commands that are part of the device property computation program. The commands are listed alphabetically; for a summary of the commands grouped according to function, see [Table 5-4](#).

Important information about how pin names and layers are used when referenced in a function argument is given in the section “[Pin and Layer References](#)” on page 1786. This section also describes how shapes are merged prior to device extraction.

## area

**AREA(*pin-or-layer*)**

### Description

Returns the total area of shapes that are part of the specified pin, or on the specified layer. The area is given in square meters.

The argument is a pin name or pin layer, as described in “[Pin and Layer References](#)” on page 1786.

## area\_common

**AREA\_COMMON(*pin-or-layer*, *pin-or-layer*)**

### Description

Returns the total area that is common to shapes on both of the pins or layers references. This value is expressed in square meters.

## bends

**BENDS(*pin-or-layer*)**

### Description

Returns the number of bends in a polygon. This is frequently useful for calculating gate bends.

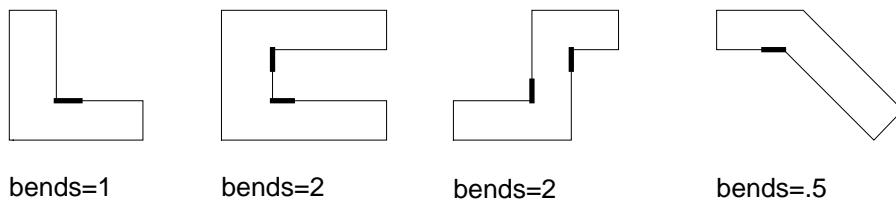
The argument is a pin name or pin layer, as described in “[Pin and Layer References](#)” on page 1786.

The function returns a floating-point number of bends that are a multiple of 90 degrees.

### Bend Calculation Examples

You can find the bends value by summing the angle, in degrees, by which the perimeter changes direction at all concave vertices, and dividing by 90 to convert to units of right angle bends. The sum is over all shapes in the specified pin or on the specified layer.

Figure 5-1 shows shapes with their concave vertices emphasized and their respective bends values indicated. Note that the change in direction of the perimeter at a vertex is not the same thing as the angle formed by the perimeter at the vertex. For example, in the rightmost shape in Figure 5-1, the angle formed by the perimeter at the indicated concave vertex is 135 degrees, but the change in perimeter direction along the perimeter is 45 degrees or 1/2 of a right angle. Thus, the bends value for this shape is 1/2.

**Figure 5-1. Computation of Bends****count****COUNT(pin-or-layer)****Description**

Returns the total number of shapes on the specified layer or in the specified pin. In device extraction, pins on a given layer are determined by the nets to which they attach, not by the number of shapes present, so it is possible for a pin to contain more than one shape. The layers are merged before device extraction, so the shapes counted do not touch or overlap.

**DFM Property Data Retrieval Functions**

The Device operation supports four data access functions that are used in conjunction with the DFM Property operation to retrieve property values for use within Device property computations. These functions are useful for performing complex geometric analysis using DFM Property operations and collecting the results of this analysis of individual extracted devices as calculated properties.

The data retrieval functions are:

**dfm\_numeric\_value** — Provides access to a DFM property from a single polygon on a pin, pin layer, or auxiliary layer. The DFM property is accessible as a regular numeric value.

**dfm\_numeric\_array** — Provides access to a DFM property from a set of polygons on a pin, pin layer, or auxiliary layer. The set of properties are accessible through a TVF (Tcl Verification Format) object created by this function.

**dfm\_vector\_value** — Provides access to a DFM vector property from a single polygon on a pin, pin layer, or auxiliary layer. The set of properties is accessible through a TVF object created by this function.

**dfm\_vector\_array** — Provides access to a DFM vector property from a set of polygons on a pin, pin layer, or auxiliary layer. The set of properties is accessible through a TVF object created by this function.

These data retrieval functions are described in the remainder of this section. The associated TVF object interfaces that access the outputs are also described.

## dfm\_numeric\_value

**DFM\_NUMeric\_VALue (*pin\_or\_layer*, “*property*”)**

### Description

The DFM\_NUMeric\_VALUE() function retrieves a single numeric property value by name from one polygon on a pin, pin layer, or auxiliary layer of the device. (The [dfm\\_numeric\\_array\(\)](#) function, allows access to multiple polygons on a layer.)

### Arguments

- ***pin\_or\_layer*** — A pin name, the name of a pin layer, or the name of an auxiliary layer in a Device operation. The layer is derived from a DFM Property operation.  
If the selected pin or layer includes more than one polygon, the device is classified as a “BAD DEVICE” in the connectivity extraction report.
- ***property*** — A property name. The property must be defined in the same DFM Property operation used for the ***pin\_or\_layer*** argument and be enclosed in quotation marks.

### Returns

A numeric property value, or the device is classified as a “BAD DEVICE” in the connectivity extraction report if the selected pin or layer includes more than one polygon.

### Examples

For example, the following Device computation creates a device property from the DFM property NC that contains the number of polygons on layer cont that interact with the S and D pins of the device:

```
// Count contacts on SD polygons and assign property "NC" to
// derived layer sd_prop polygons.
sd_prop = DFM PROPERTY sd cont [ NC = COUNT(cont) ]

// Produce device properties nc_s and nc_d based on DFM property "NC"
// for S and D pins.
DEVICE MP pgate poly(G) sd_prop(S) sd_prop(D) [
    property nc_s, nc_d
    nc_s = DFM_NUMeric_VALue( S, "NC" )
    nc_d = DFM_NUMeric_VALue( D, "NC" )
]
```

Note that when multiple polygons qualify for selection by the DFM\_NUMeric\_VALUE() function, the device is classified as bad even if the DFM\_NUMeric\_VALUE() function is never executed due to conditional statements. For example:

```
// Produce device property nc_aux
DEVICE MP pgate poly(G) sd(S) sd(D) <aux1> [
    property nc_aux
    if (COUNT(aux1) == 1 ) {
        // This conditional protection does not prevent the device from being
        // classified as a bad device in cases where this condition fails.
        nc_aux = DFM_NUMeric_VALue( aux1, "NC" )
    } else {
        nc_aux = 0
    }
]
```

```

        }
    ]

```

In cases where there may be more than one polygon associated with the pin or layer, always use the DFM\_NUMERIC\_ARRay() function to access the associated DFM properties.

DFM\_NUMERIC\_VALUE() supports DFM NetID properties. NetID properties act similarly to numeric properties within the Device computation language. For example, the following [DFM Property](#) and Device program produce Net IDs similar to the PIN\_NET function for source and drain pins in a MOSFET device:

```

sd = diff not poly
sd_prop = DFM PROPERTY sd [ NET = NETID( sd ) ]

device mp pgate poly(G) sd_prop(S) sd_prop(D) well(B)
[
    PROPERTY src_net_prop, drn_net_prop, src_pin_net, drn_pin_net
    src_net_prop = dfm_num_val( s, "NET" )
    drn_net_prop = dfm_num_val( d, "NET" )
    src_pin_net = pin_net( s )
    drn_pin_net = pin_net( d )
]

```

## dfm\_numeric\_array

### [DFM\\_NUMERIC\\_ARRay \(pin-or-layer, “property”\)](#)

#### Description

The DFM\_NUMERIC\_ARRay() function retrieves numeric property values by name from one or more polygons on a pin, pin layer, or auxiliary layer of the device. The DFM\_NUMERIC\_ARRay() function is similar to the [dfm\\_numeric\\_value\(\)](#) function, but includes the ability to handle more than one polygon at a time.

#### Arguments

- ***pin\_or\_layer*** — A pin name, the name of a pin layer, or the name of an auxiliary layer in a Device operation. The layer is derived from a DFM Property operation.
- ***property*** — A property name. The property must be defined in the same DFM Property operation used for the ***pin\_or\_layer*** argument and be enclosed in quotation marks.

#### Returns

The DFM\_NUMERIC\_ARRay() function returns a Tcl object as described in [“TVF Object for dfm\\_numeric\\_array\(\)”](#) on page 1790. Values from the Tcl object must be accessed using [TVF\\_numeric\\_function](#).

## TVF Object for dfm\_numeric\_array()

The DFM\_NUMERIC\_ARRay() function returns a Tcl object that provides access to DFM property data found on individual shapes that make up a device. The object represents a set of polygons, each of which has a numeric value associated with it.

The object supports the following methods, where ***object*** refers to the Tcl object produced by the DFM\_NUMERIC\_ARRAY() function:

- ***object polygon\_count*** — Returns the number of polygons associated with the pin, pin layer, or auxiliary layer referenced by the DFM\_NUMERIC\_ARRAY() function.
- ***object value polygon\_number*** — Returns the value of the DFM property associated with the polygon having the ***polygon\_number***. The ***polygon\_number*** must evaluate to a non-negative integer less than ***polygon\_count***.
- ***object bad\_device*** — Flags a device as bad to the Calibre Device Extraction system. After a device is flagged as bad it is handled similarly to other bad devices (that is, devices missing pin interactions, and so forth.)

You must include this statement

```
package require CalibreLVS_DEVICE_DFM
```

in any TVF function block that includes a Tcl proc which references a DFM\_NUMERIC\_ARRAY() object.

#### Example

For example, the following Device computation creates a device property nc from the DFM property NC, which contains the number of polygons on layer cont that interact with both sides of the sd layer adjacent to the gate polygon.

The DFM\_NUMERIC\_ARRAY() function creates the nc\_tcl Tcl object, which gets passed to the Tcl proc named total\_count. The function [TVF\\_NUMERIC\\_FUNCTION](#) is used to retrieve the value returned by Tcl proc total\_count. (Also see “[TVF Functional Interface for Device Property Calculation](#)” on page 1827.)

```
// Count contacts on sd polygons and assign property "NC" to
// derived layer sd_prop polygons.
sd_prop = DFM PROPERTY sd cont [ NC = COUNT(cont) ]

// Produce a device property "nc" based on DFM property "NC"
DEVICE MP gate gate(G) sd_prop(S) sd_prop(D) [
    property nc
    nc_tcl = dfm_num_arr( sd_prop, "NC" )
    nc = tvf_num_fun::device_func::total_count( nc_tcl )
]

// Define the tcl proc total_count in a TVF Function block
TVF FUNCTION device_func /*
package require CalibreLVS_DEVICE_DFM

// total_count sums the properties across all input polygons and returns
// the total. Any device that includes more than two polygons on this
// device layer is marked as bad
proc total_count { dfm_array } {
    set pc [ $dfm_array polygon_count ]
    if { $pc > 2 } {
        $dfm_array bad_device
        return 0.0
    }
}
```

```

set accum 0.0
for { set i 0 } { $i < $pc } { incr i } {
    set accum [ expr { $accum + [ $dfm_array value $i ] } ]
}
return $accum
}
*/]

```

## [dfm\\_vector\\_value](#)

**DFM\_VECtor\_VALue (*pin-or-layer*, “*property*”)**

### Description

The DFM\_VECtor\_VALue() function provides access to DFM vector properties (such as are generated by the DFM Property VECTOR or VPROPERTY functions). DFM vector properties are arranged as a sequence of measurement tuples. This function provides access to the DFM vector property from a single polygon associated with a device pin, pin layer, or auxiliary layer. (The [dfm\\_vector\\_value\(\)](#) function allows access to vector properties from multiple polygons, and is described later in this section.)

### Arguments

- ***pin\_or\_layer*** — A pin name, the name of a pin layer, or the name of an auxiliary layer in a Device operation. The layer is derived from a DFM Property operation.  
If more than one polygon corresponds to the call, the device is classified as a “BAD DEVICE” in the circuit extraction report.
- ***property*** — A property name. The property must be defined in the same DFM Property operation used for the ***pin\_or\_layer*** argument and be enclosed in quotation marks.

### Returns

The DFM\_VECtor\_VALue() function returns a Tcl object. The values can be read in these ways:

- Using [TVF\\_numeric\\_function](#) to access the values, as described in [TVF Object for dfm\\_vector\\_value\(\)](#) and [Example Using TVF Object for dfm\\_vector\\_value\(\)](#).
- Using the device TVF functions `device::dfm_vec_measurements` and `device::scaled_dfm_vec_measurements`, described in [Using Device TVF Functions to Read dfm\\_vector\\_value\(\) Objects](#).

If more than one polygon corresponds to the call, the device is classified as a “BAD DEVICE” in the circuit extraction report.

## [TVF Object for dfm\\_vector\\_value\(\)](#)

The [dfm\\_vector\\_value\(\)](#) function returns a Tcl object that provides access to DFM Property vector data found on a particular shape of a device. The DFM vector properties consist of an array of measurement tuples that are associated with a particular shape. These properties are, in turn, transferred from a device pin, pin layer, or auxiliary layer to the SVRF Device property

computation program through a Tcl command object. The Tcl command object provides methods for determining the number of measurements represented by the referenced layer or pin, for providing the dimensions of the DFM Property vector, and for accessing individual values by vector index coordinates.

Throughout this section, ***object*** refers to the Tcl object produced by the DFM\_VECtor\_VALue() function. The DFM\_VECtor\_VALue() object has the following methods:

- ***object row\_count*** — Returns the number of measurement rows or measurement tuples associated with the DFM vector property. Returns a non-negative integer.
- ***object measurement\_count*** — Returns the dimension of the measurement tuples associated with the DFM vector property. Returns a non-negative integer.
- ***object value row measurement*** — Returns the value at the ***row measurement*** location of the DFM vector property. The ***row*** and ***measurement*** arguments are expected to evaluate to non-negative integers.

#### Example Using TVF Object for dfm\_vector\_value()

For example, the following Device computation creates a device property, p\_res, from the DFM property RES that is constructed on the gate pin.

```
gate_p = DFM PROPERTY gate g2s OVERLAP ABUT ALSO MULTI
    [ RES = RANGE_XXY( SORT_MERGE_XXY(
        VECTOR( ECMIN(g2s), ECMAX(g2s), EW(g2s) )
        ,
        VECTOR( ECMIN(g2s), ECMAX(g2s) )
    )
)
]

DEVICE MP gate gate_p(G) sd(S) sd(D) [
    property p_res
    p_res_tcl = DFM_VECtor_VALue( G, "RES" )
    p_res = TVF_numeric_function::device_func::get_sum( p_res_tcl )
]
```

The p\_res\_tcl argument is a Tcl object generated by DFM\_VECtor\_VALue(), which gets passed to the Tcl proc named get\_sum. The Tcl proc sums the properties across all input polygons and returns them to the **TVF\_numeric\_function** that called the proc in the Device computation program:

```
TVF FUNCTION device_func /* 
package require CalibreLVS_DEVICE_DFM

proc get_sum { dfm_vector } {
    set meas_count [ $dfm_vector measurement_count ]
    set row_count [ $dfm_vector row_count ]
    set accum 0.0
    for { set i 0 } { $i<$row_count } { incr i } {
        for {set j 0} { $j<$meas_count } { incr j } {
            set accum [expr { $accum + [ $dfm_vector value $i $j ] }]
        }
    }
}
```

```

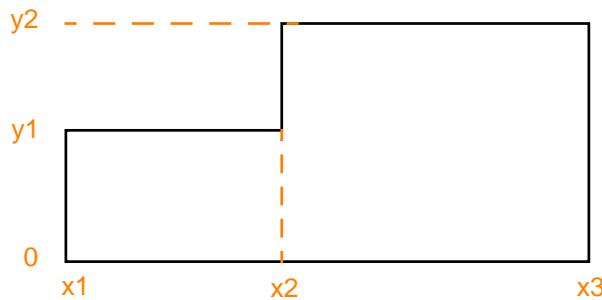
        return $accum
    }
*/ ]

```

## Using Device TVF Functions to Read dfm\_vector\_value() Objects

The Device TVF language provides two additional functions, device::dfm\_vec\_measurements and device::scaled\_dfm\_vec\_measurements, that are useful in accessing data from a dfm\_vector\_value object. These functions provide a convenient access method for DFM Property vectors constructed using the SORT\_MERGE\_XXY and SELECT\_MERGE\_XXY DFM Property functions. The device::dfm\_vec\_measurements and device::scaled\_dfm\_vec\_measurements functions take a DFM Property vector returned from the DFM\_VECtor\_VALue() function and return a list of measurements that are similar to those provided by the Device enclosure operations (see [ENClosure\\_PARallel and ENCclosure\\_PERpendicula Functions](#)).

The DFM Property SORT\_MERGE\_XXY and SELECT\_MERGE\_XXY functions return a set of coordinate vectors that outline a group of segmented measurement regions where coordinates trace the outlines of measured areas as shown in this figure.



The SORT\_MERGE\_XXY function provides a rectilinear measurement vector that describes this region as follows:

```
{ (x1,y1), (x2,y2), (x3,0) }
```

However, the Device enclosure functions provide measurements that are a sequence of width and length measurements, such as these:

$$w_1 = x_2 - x_1$$

$$w_2 = x_3 - x_2$$

These measurements are then arranged in a vector that describes this region as follows:

```
{ (w1,y1), (w2,y2) }
```

Both methods are valid for describing the region, and the method you choose is simply a matter of preference.

Note the DFM Property outputs are in database units while the Device enclosure operations are in user units.

The functions for converting the dfm\_vector\_value object to enclosure vector format are these:

- **device::dfm\_vec\_measurements *object*** — converts DFM-style rectilinear measurement vectors into Device enclosure-vector-style width and length measurements. The *object* is a DFM\_VECTor\_VALue() object. The measurements remain in database units.
- **device::scaled\_dfm\_vec\_measurements *object scale\_factor*** — converts DFM-style rectilinear measurement vectors into Device enclosure-vector-style width and length measurements. The *object* is a DFM\_VECTor\_VALue() object. The *scale\_factor* is used to scale the results to dimensions other than user units. Output from the *unit\_length()* function in the Device property computation language can be used to scale database units to user units.

The example code in step 3 in the section “[Performing Well Enclosure or Stress Effect Calculations with DFM Property](#)” on page 1837 shows a use of the device::scaled\_dfm\_vec\_measurements function.

These functions are available in a Tcl package called CalibreLVS\_DEVICE\_DFM. In order to access these functions, you must have the following command in your TVF FUNCTION block:

```
package require CalibreLVS_DEVICE_DFM
```

The functions in this package are in the device:: namespace.

## dfm\_vector\_array

**DFM\_VECTor\_ARRay (*pin-or-layer*, “*property*”)**

### Description

The DFM\_VECTor\_ARRay() function accesses a set of DFM vector properties (such as those generated by the DFM Property VECTOR or VPROPERTY functions). DFM vector properties are arranged as a sequence of measurement tuples. They are arranged by name from *multiple* polygons on a pin, pin layer, or auxiliary layer of the device. The function is similar to [dfm\\_vector\\_value\(\)](#) except that it is not limited to a single polygon.

### Arguments

- ***pin\_or\_layer*** — A pin name, the name of a pin layer, or the name of an auxiliary layer in a Device operation. The layer is derived from a DFM Property operation.
- ***property*** — A property name. The property must be defined in the same DFM Property operation used for the ***pin\_or\_layer*** argument and be enclosed in quotation marks.

### Returns

The DFM\_VECTor\_ARRay() function returns a Tcl object described in [TVF Object for dfm\\_vector\\_array\(\)](#).

## TVF Object for dfm\_vector\_array()

The DFM\_VECtor\_ARRay() function returns a Tcl object that represents a set of individual polygons, each of which has a DFM Property vector value associated with it. The DFM Property operation produces an array of measurement tuples that are associated with a particular shape. These properties are in turn transferred from a device pin, pin layer, or auxiliary layer to the SVRF Device property computation program by the DFM\_VECtor\_ARRay() function in the Device built-in language.

The Tcl object provides methods for determining the number of polygons represented by the referenced layer or pin, determining the dimensions of the DFM Property vector for a particular polygon, and accessing individual values by polygon and vector index coordinates. It also provides a method by which a device can be flagged to the Calibre device extraction system as a bad device.

Throughout this section, ***object*** refers to the Tcl object produced by the DFM\_VECtor\_ARRay() function. The following methods are provided for the DFM\_VECtor\_ARRay() object:

- ***object polygon\_count*** — Returns the number of polygons associated with the pin, pin layer, or auxiliary layer referenced by the DFM\_VECtor\_ARRay() function.
- ***object row\_count polygon\_number*** — Returns the number of measurement rows or measurement tuples in the DFM vector property associated with ***polygon\_number***. The ***polygon\_number*** must evaluate to a non-negative integer less than ***polygon\_count***.
- ***object measurement\_count polygon\_number*** — Returns the dimension of the measurement tuples in the DFM vector property associated with ***polygon\_number***. The ***polygon\_number*** must evaluate to a non-negative integer less than ***polygon\_count***.
- ***object value polygon\_number row measurement*** — Returns the value of the DFM vector property associated with the polygon having the ***polygon\_number***, at the ***row measurement*** location of the DFM vector property. The ***row*** and ***measurement*** arguments are expected to evaluate to non-negative integers. The ***polygon\_number*** must evaluate to a non-negative integer less than ***polygon\_count***.
- ***object bad\_device*** — Flags a device as bad to the Calibre Device Extraction system. After a device is flagged as bad it is handled similarly to other bad devices (that is, devices missing pin interactions, and so forth.)

### Example

For example, the following Device computation creates a device property, sum\_res, from the DFM property RES that is constructed on the aux\_prop auxiliary layer shape.

A tcl proc get\_sum is defined in a [TVF Function](#) block to process the Tcl object output by the DFM\_VECtor\_ARRay() function. The get\_sum proc sums the properties across all input polygons. Any device that includes more than two polygons on this device layer is marked as bad.

```
aux_prop = DFM PROPERTY aux1 g2s OVERLAP ABUT ALSO MULTI
[ RES = RANGE_XXY( SORT_MERGE_XXY(
```

```

        VECTOR( ECMIN(g2s), ECMAX(g2s), EW(g2s) )
    ),
    VECTOR( ECMIN(g2s), ECMAX(g2s) )
)
]

DEVICE MP gate gate(G) sd(S) sd(D) <aux_prop> [
    property sum_res
    sum_res_tcl = DFM_VECToR_ARRAy( aux_prop, "RES" )
    sum_res = TVF_numeric_function::device_func::get_sum( sum_res_tcl )
]

// Define the tcl proc get_sum in a TVF Function block
TVF FUNCTION device_func /*[
package require CalibreLVS_DEVICE_DFM

proc get_sum { dfm_vec_array } {
    set pc [ $dfm_vec_array polygon_count ]
    if { $pc > 2 } {
        $dfm_vec_array bad_device
        return 0.0
    }
    set accum 0.0
    for { set p 0 } { $p < $pc } { incr p } {
        set meas_count [ $dfm_vec_array measurement_count $p ]
        set row_count [ $dfm_vec_array row_count $p ]
        for { set i 0 } { $i < $row_count } { incr i } {
            for {set j 0} { $j < $meas_count } { incr j } {
                set accum [ expr { $accum + [ $dfm_vec_array value $p $i $j ] }
            }
        }
    }
    return $accum
}
*/]

```

## device::dfm\_vec\_measurements

**device::dfm\_vec\_measurements object**

### Description

Converts DFM-style rectilinear measurement vectors into Device enclosure-vector-style width and length measurements. The measurements remain in database units.

The device::dfm\_vec\_measurements function is used in conjunction with the [TVF Functional Interface for Device Property Calculation](#). It must be used in a TVF FUNCTION block. See “[Using Device TVF Functions to Read dfm\\_vector\\_value\(\) Objects](#)” on page 1794 for a complete explanation of the function.

### Arguments

- *object* — A dfm\_vector\_value object.

## Related Topics

Also see [device::scaled\\_dfm\\_vec\\_measurements](#).

## device::enclosure\_measurements

```
device::enclosure_measurements -base layer -measurement layer -orient layer [-connect]
[-max_par value] [-max_per value] [-help | ?]
```

### Description

The device::enclosure\_measurements function is used in conjunction with the [TVF Functional Interface for Device Property Calculation](#). This is a Tcl function and must be used in a TVF FUNCTION block. This function has performance advantages over other device enclosure functions that were introduced in releases prior to 2008.3. It is the recommended function for performing well-proximity enclosure and stress effect calculations.

This function provides the ability to take similar measurements to ENCclosure\_PERpendicular and ENCclosure\_PARallel, which are discussed under [“ENCclosure\\_PARallel and ENCclosure\\_Perpendicular Functions”](#) on page 1807. Also see the section [“Comparison of Enclosure Calculation Functions and Methods”](#) on page 1833 for further details of other enclosure functions.

The device::enclosure\_measurements function is a part of the CalibreDFM\_DEVICE package and requires a Calibre Advanced Device Properties (ADP) product license.

This implementation has performance advantages for well enclosure calculations, especially for well layers derived from the [Extent](#) operation and having a large number of vertices.

### Arguments

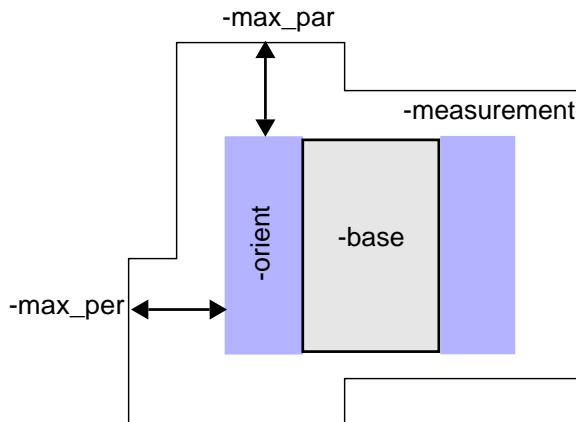
- **-base *layer*** — a required argument that defines the layer from which to measure the enclosure. Usually this is the gate layer of a MOS transistor.
- **-measurement *layer*** — a required argument that defines the layer that encloses the **-base** layer. Usually the **-measurement** layer is a well layer.
- **-orient *layer*** — a required argument that defines an orientation layer. Usually this is a source/drain layer.
- **-connect** — an option that specifies the output layer should include PER\_NETID\_<N> properties. This option is used when verifying that the measurements taken by device::enclosure\_measurements correlate with the same pins as the measurements taken by the Device property computation. If the **-connect** option is used, connectivity must be established on the **-orient** layer.
- **-max\_par *value*** — an option that defines the maximum parallel-oriented search distance in user units from a **-base** layer edge to an enclosing **-measurement** layer edge. The *value* must evaluate to a number. This option overrides any global specification of this search distance. The default *value* is -1.0, which indicates the option is not set.
- **-max\_per *value*** — an option that defines the maximum perpendicular-oriented search distance in user units from a **-base** layer edge to an enclosing **-measurement** layer edge.

The *value* must evaluate to a number. This option overrides any global specification of this search distance. The default *value* is -1.0, which indicates the option is not set.

- -help | ? — prints the usage message for this function in a Tcl shell.

Any errors in specifying the function are reported at compile time. [Figure 5-2](#) shows the parameters with their associated layers and measurement distances.

**Figure 5-2. device::enclosure\_measurements Parameters**



In a standard application of device::enclosure\_measurements, the **-base** layer contains MOS gates, the **-measurement** layer is a MOS well layer, and the **-orient** layer is a MOS source/drain layer.

The **-max\_par** and **-max\_per** arguments are optional, but must be specified in the global scope if not specified as options to the function. To set the search distances (in user units) in the global scope, specify the following variables using your own search distances:

```
set device::max_per <value>
set device::max_par <value>
```

These settings apply to all instances of device::enclosure\_measurements functions unless overridden with the local function arguments. The distance parameters must evaluate to numbers. Because global variables are easier to maintain than local arguments, use the global settings to set search distances unless you must override the value for one type of device by using a local value.

Each search distance must be specified explicitly in the rule file. There are no default values contained in the TVF package. You may set any of the search distances to 0 if the process allows it. Any missing search distances result in a compile-time error.

You may not use the [Variable](#) statement to set the Tcl program's search distance variables, either globally or locally.

For example, this is not allowed:

```
VARIABLE MAX_PAR 3
...
set device::max_par MAX_PAR; # bad. cannot set this value from a Variable.
```

```
tvf::SETLAYER prop = device::enclosure_measurements
...
-max_par MAX_PAR; # bad. cannot set this value from a variable.
```

This is allowed:

```
set device::max_par 3
tvf::SETLAYER prop = device::enclosure_measurements
...
-max_par 3
```

You may use valid Tcl expressions to set search distance variables. For example:

```
set device::max_par 3
set device::max_per [ expr { $device::max_par * 2 } ]

set myvar 12
tvf::SETLAYER prop = device::enclosure_measurements
...
-max_par [ expr { $myvar * 2 } ] -max_per $myvar
```

The output of device::enclosure\_measurements is a *layer* to be specified as an auxiliary layer (specified in angle brackets: <*layer*>) in a Device statement. The output layer may not be used as a seed layer in a Device statement. The output layer need not be derived uniquely for each specialized device seed. The device::enclosure\_measurements function can be used with the most general predecessor of the actual device layers. For example, if you derive your gate layer this way:

```
gate = poly and diff
```

and gate is the predecessor of all gate layers in the design, then this layer can be used for device::enclosure\_measurements function to generate the auxiliary layer. This can simplify the rule file.

The device::enclosure\_measurements output layer contains DFM property annotations. These properties are available from the Device program through the DFM\_\* vector functions ([“DFM Property Data Retrieval Functions”](#) on page 1788), and can be used by your custom Tcl procs to compute an effective property to netlist.

Each polygon on the device::enclosure\_measurements output layer includes four DFM vector objects:

- PER\_1 — Enclosure vector measured perpendicular to gate length corresponding to the first source/drain shape.
- PAR\_1 — Enclosure vector measured parallel to gate length corresponding to the first source/drain shape.
- PER\_2 — Enclosure vector measured perpendicular to gate length corresponding to the second source/drain shape.
- PAR\_2 — Enclosure vector measured parallel to gate length corresponding to the second source/drain shape.

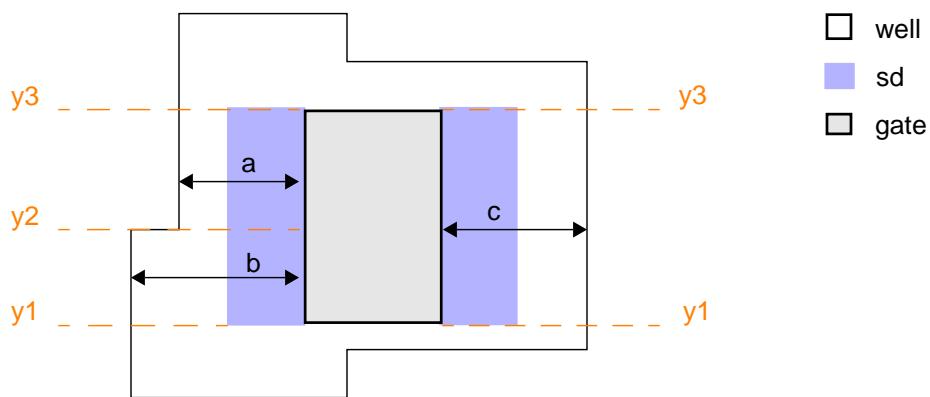
The four enclosure vectors are of arbitrary length. There is one segment for each jog in either the base or the measurement polygons. Each member of the enclosure vector contains two values:

- the first coordinate of the segment
- the enclosure distance for the segment

The final member of the enclosure vector has this form: final coordinate of segment, 0.

[Figure 5-3](#) shows an example of the perpendicular measurements.

**Figure 5-3. device::enclosure\_measurements Parameters**



Given the preceding diagram, the perpendicular vectors would have these forms.

```
PER_1 <y1,b> <y2,a> <y3,0>
PER_2 <y1,c> <y3,0>
```

The parallel measurements would be made in the vertical direction and the PAR\_1 and PAR\_2 vectors would have similar forms to the ones shown here.

Vector elements are reported in user units. You must use the [unit\\_length\(\)](#) function to convert to meters in order to get appropriately-valued device properties into the extracted netlist. This is shown in the example later in this section.

To assist in obtaining the desired information from the enclosure vectors, the CalibreLVS\_DEVICE\_DFM TVF package (which is included in the software tree and gets called automatically as needed), includes some helper functions for processing DFM vector objects. The measurement operations included in this package have been optimized for performance.

A useful function from this package is [device::dfm\\_vec\\_measurements](#). This function converts the DFM enclosure vectors into Tcl lists of {width value} segments, where width is the length of a segment and value is the enclosure distance. From the preceding example, we have these vectors:

```
PER_1 <y1,b> <y2,a> <y3,0>
PER_2 <y1,c> <y3,0>
```

Evaluating these vectors using device::dfm\_vec\_measurements produces the desired {width value} lists. For PER\_1, one {width value} list would be {y2-y1 b}, and the other would be {y3-y2 a}. For PER\_2, the {width value} list would be {y3-y1 c}.

The product of width × value gives an enclosure area for one segment of the gate edge. This is used in the example later in this section.

The CalibreDFM\_DEVICE package, which is also in the software tree, provides two variables to maximize the optimization for your layout style:

- device::avg\_sp3 — Average width of a box drawn around three measurement polygons. Default is 0.5 user units.
- device::avg\_enc — Average measurement polygon to base polygon enclosure distance. Default is 1.5 user units.

The values need not be exact. It is usually sufficient to use one set of values for each process technology. There may be performance penalties if the settings are either too large or too small for the technology. The default settings are targeted for 45nm SOC designs.

Here is an example:

```
#!tvf
tvf::VERBATIM {
    TVF FUNCTION mytcl [/*
        # This package is required for device::enclosure_measurements.
        package require CalibreLVS_DEVICE_DFM

        # This proc computes average enclosure values. The dfmvec argument
        # is a property vector passed from the mos_prop macro.
        proc avg_dfm_vec { dfmvec } {

            # $dfmvec contains vectors from which {width value} lists are
            # calculated using the following line.
            set measurements [ device::dfm_vec_measurements $dfmvec ]
            # sum is a sum of enclosure areas.
            set sum 0.0
            # tot is the total length of a gate edge.
            set tot 0.0

            foreach segment $measurements {
                foreach { width value } $segment {
                    puts "dfm $width $value"
                    set sum [ expr { $sum + $value * $width } ]
                    set tot [ expr { $tot + $width } ]
                }
            }
            # This returns an average enclosure distance for a gate edge.
            return [ expr { $sum / $tot } ]
        }

    /*]
// end TVF FUNCTION
```

```

// This macro defines four enclosure measurement properties
// for each device. The PAR_1, PAR_2, PER_1, and PER_2 objects
// are stored with the layers ngate_prop, and pgate_prop, which
// are passed in as the prop_layer parameter.

DMACRO mos_prop prop_layer {
    [
    PROPERTY perA, perB, parA, parB
    // get the four vectors from the vector property layers.
    par_dfm_A = dfm_vector_value(prop_layer, "PAR_1")
    par_dfm_B = dfm_vec_val(prop_layer, "PAR_2")
    per_dfm_A = dfm_vec_val(prop_layer, "PER_1")
    per_dfm_B = dfm_vec_val(prop_layer, "PER_2")
    // calculate the average enclosure values on the four sides
    // of the gate. call the avg_dfm_vec proc to do this.
    parA = TVF_numeric_function::mytcl::avg_dfm_vec(par_dfm_A) *
unit_length()
    parB = tvf_num_fun::mytcl::avg_dfm_vec(par_dfm_B) * unit_length()
    perA = tvf_num_fun::mytcl::avg_dfm_vec(per_dfm_A) * unit_length()
    perB = tvf_num_fun::mytcl::avg_dfm_vec(per_dfm_B) * unit_length()
    ]
}
}

// These are the device statements. The auxiliary layers ngate_prop and
// pgate_prop are derived from device::enclosure_measurements functions.
// The mos_prop macro defines the four enclosure measurement
// properties for each device.

DEVICE mn ngate nsd nsd sub <ngate_prop> CMACRO mos_prop ngate_prop
DEVICE mp pgate psd psd nw <pgate_prop> CMACRO mos_prop pgate_prop

}

tvf:// end VERBATIM block

tvf:// The following package is required.
    package require CalibreLVS_DEVICE_DFM

tvf:// Globally set the maximum perpendicular and parallel enclosure
tvf:// distances.

    set device::max_par 15.0
    set device::max_per 3.0

tvf:// Define the auxiliary layers for the Device statements.

tvf::SETLAYER ngate_prop = [ device::enclosure_measurements -base ngate
-measurement psub -orient nsd ]

tvf::SETLAYER pgate_prop = [ device::enclosure_measurements -base pgate
-measurement nwell -orient psd ]

```

When the -connect option is used, each output polygon includes two DFM numeric properties in additions to the four vector properties discussed previously. The two numeric properties are these:

- PER\_NETID\_1 — Contains the net ID of the shape for which PER\_1 is measured.
- PER\_NETID\_2 — Contains the net ID of the shape for which PER\_2 is measured.

The net ID values stored by these properties may be retrieved using the `dfm_numeric_value()` function and compared to what the `pin_net()` function returns. This is done to correlate the `device::enclosure_measurements` computations with the Device operation's computations on the same pin. This ensures that the measurements refer to the same pin.

Here is an example:

```
#!tvf
...
tvf::SETLAYER ENC_PROP = [ device::enclosure_measurements -base gate \
                           -orient sd -measurement well -connect ]
tvf::VERBATIM {
  DEVICE MN gate poly sd sd well <ENC_PROP> [
    PROPERTY AS, AD, ES, ED
    AS = AREA(S)
    AD = AREA(D)
    EV_1 = DFM_VEC_VAL(ENC_PROP, "PER_1")
    E_1 = TVF_NUM_FUN::devlib::avg(EV_1)
    EV_2 = DFM_VEC_VAL(ENC_PROP, "PER_2")
    E_2 = TVF_NUM_FUN::devlib::avg(EV_2)
    N_1 = DFM_NUM_VAL(ENC_PROP, "PER_NETID_1")
    N_2 = DFM_NUM_VAL(ENC_PROP, "PER_NETID_2")

    IF (PIN_NET(S) == N_1) { ES = E_1 ED = E_2 }
    ELSE IF (PIN_NET(S) == N_2) { ES = E_2 ED = E_1 }
    ELSE { ES = -1 ED = -1 }
  ]
}
tvf:// end VERBATIM block
```

The final IF block is used to correlate the `device::enclosure_measurements` computations with the Device operation's computations. The function called `devlib` and the proc called `avg` would be defined in a TVF FUNCTION block.

## device::eval\_dfm\_func

**device::eval\_dfm\_func *dfm\_func\_name parameter\_list***

### Description

The device::eval\_dfm\_func function evaluates DFM Function TABLE and MATRIX functions defined in the rule file.

This function is available in a Tcl package called CalibreLVS\_DEVICE\_DFM. In order to access this function, you must have the following command in your TVF FUNCTION block:

```
package require CalibreLVS_DEVICE_DFM
```

The functions in this package are in the device:: namespace.

### Arguments

- *dfm\_func\_name* — The name of a DFM Function TABLE or MATRIX function.
- *parameter\_list* — A list of parameters to the DFM Function.

### Returns

A numeric value.

### Example

For example, consider the following DFM Function statements:

```
DFM FUNCTION [ dfm_f0( number x )
    TABLE { 0.0 1.32e-06 1e-06 1.25e-06 2e-06 0.5e-06 }
]

DFM FUNCTION [ dfm_f1( number x1, number x2, number x3 )
    MATRIX TRUNCATE
        { 0.0, 2e-06 }
        { 0.0, 2e-06 }
        { 0.0, 2e-06 }
        ::
        { 2.3, 2.5 }
        { 2.8, 2.9 }
        { 3.1, 3.2 }
        { 2.4, 2.2 }
]
```

In the first statement, the values that x can take are within the range of values 0.0, 1e-06, and 2e-06. The values the function returns are within the range of values 1.32e-06, 1.25e-06, and 0.5e-06. Depending upon what x is, within the range of the three corresponding values of the table, an interpolated value is returned from the range of return values.

In the second statement, the values x1, x2, and x3 can take are shown in the first three lists, respectively. They are 0.0 and 2e-06 in each case. This table shows the return values based upon possible x1, x2, and x3 values.

<b>x1, x2, x3</b>	<b>Return value</b>
0.0, 0.0, 0.0	2.3

<b>x1, x2, x3</b>	<b>Return value</b>
2e-06, 0.0, 0.0	2.5
0.0, 2e-06, 0.0	2.8
2e-06, 2e-06, 0.0	2.9
0.0, 0.0, 2e-06	3.1
2e-06, 0.0, 2e-06	3.2
0.0, 2e-06, 2e-06	2.4
2e-06, 2e-06, 2e-06	2.2

These can be called from inside a Device TVF Tcl procedure:

```
TVF FUNCTION device_func [/*
package require CalibreLVS_DEVICE_DFM

proc use_dfm_func_f0 { a } {
    return [ device::eval_dfm_func "dfm_f0" [$a] ]
}
proc use_dfm_func_f1 { a b c } {
    return [ device::eval_dfm_func "dfm_f1" "[ $a ] [ $b ] [ $c ]" ]
}
*/]
```

which, in turn, can be called from a Device property computation program using [TVF\\_numeric\\_function](#), as follows:

```
DEV MN ngate ngate sd sd bulk [
    property p_f0, p_f1
    w = perimeter_coincide( sd, ngate )/2
    l = area( ngate )/w

    p_f0 = tvf_num_fun::device_func::use_dfm_func_f0( w )
    p_f1 = tvf_num_fun::device_func::use_dfm_func_f1( w, l, p_f0 )
]
```

## device::scaled\_dfm\_vec\_measurements

**device::scaled\_dfm\_vec\_measurements object scale\_factor**

### Description

Converts DFM-style rectilinear measurement vectors into Device enclosure-vector-style width and length measurements.

The device::scaled\_dfm\_vec\_measurements function is used in conjunction with the [TVF Functional Interface for Device Property Calculation](#). It must be used in a TVF FUNCTION block. See “[Using Device TVF Functions to Read dfm\\_vector\\_value\(\) Objects](#)” on page 1794 for a complete explanation.

## Arguments

- *object* — A dfm\_vector\_value object.
- *scale\_factor* — Used to scale the results to dimensions other than user units. Output from the [unit\\_length\(\)](#) function in the Device property computation language can be used to scale database units to user units.

## Related Topics

Also see [device::dfm\\_vec\\_measurements](#).

## ENClosure\_PARallel and ENCclosure\_PERpendicuLar Functions

These functions can be used for measuring enclosure distances.

For well enclosure measurement applications, these functions have largely been replaced by [device::enclosure\\_measurements](#) because it is more efficient.

[Table 5-8](#) shows differences in the methods employed by these functions.

There are four functions divided between parallel and perpendicular measurement orientations, and between non-multifinger and multifinger device configurations, as listed here:

- enclosure\_parallel
 

**ENClosure\_PARallel (*base-pin-or-layer*, *measurement-pin-or-layer*,  
*orientation-pin-or-layer*, *constraint-distance*)**
- enclosure\_parallel\_multifinger
 

**ENClosure\_PARallel\_MULTifinger (*base-pin-or-layer*,  
*measurement-pin-or-layer*, *orientation-pin-or-layer*, *constraint-distance*)**
- enclosure\_perpendicular
 

**ENClosure\_PERpendicuLar (*base-pin-or-layer*, *measurement-pin-or-layer*,  
*orientation-pin-or-layer*, *constraint-distance*)**
- enclosure\_perpendicular\_multifinger
 

**ENClosure\_PERpendicuLar\_MULTifinger (*base-pin-or-layer*,  
*measurement-pin-or-layer*, *orientation-pin-or-layer*, *constraint-distance*)**

These functions are described together since they share common concepts and parameters. The following sections are included:

[Description](#) — Description of the measurements made.

[Arguments](#) — Definition of the arguments.

[Enclosure Arrays and Operators](#) — Description of the output array.

[Restrictions](#) — Restrictions on the input parameters, and how to use the output enclosure array.

[Licensing](#) — What licenses are used.

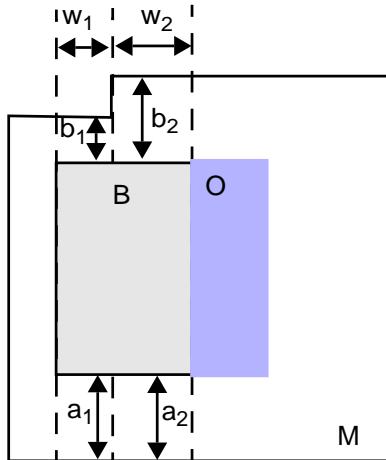
[Debugging](#) — Use of the DEBUG statement.

See “[Well Proximity Property Computation Examples](#)” on page 1834 for examples using these functions.

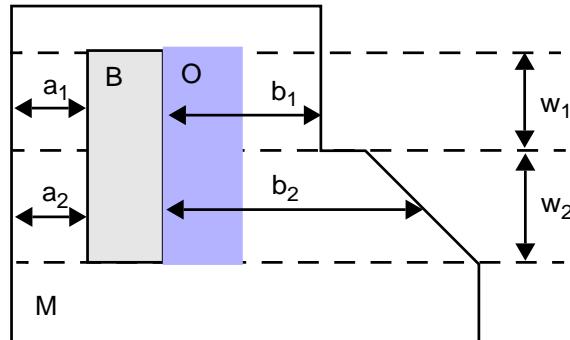
### Description

In the remainder of this description, the *base-pin-or-layer* is referred to as  $B$ , the *measurement-pin-or-layer* is referred to as  $M$ , and the *orientation-pin-or-layer* is referred to as  $O$ . Refer to Figure 5-4 and 5-5.

**Figure 5-4. ENClosure\_PARallel() and ENCclosure\_PARallel\_MULTifinger()**



**Figure 5-5. ENCclosure\_PERpendicuLar() and ENCclosure\_PERpendicuLar\_MULTifinger()**



Note that the measurements  $a$ ,  $b$ , and  $w$  are not always oriented in the same way versus the device seed shape (left-to-right or bottom-to-top, and so forth). However, the tool will calculate the vectors containing these measurements consistently.

Each function measures a set of values regarding the enclosure of  $M$  around  $B$ .

`ENCclosure_PARallel()` and `ENCclosure_PARallel_MULTifinger()` measure enclosures in the direction parallel to common edges of  $B$  and  $O$ . `ENCclosure_PERpendicuLar()` and `ENCclosure_PERpendicuLar_MULTifinger()` measure enclosures in the direction perpendicular to the common edges of  $B$  and  $O$ .

For these functions, the following conditions are enforced:

- In the non-multifinger functions, there must be exactly one polygon on the base layer **B** touching or intersecting the device seed.
- In the multifinger functions, multiple **B** polygons are allowed. In either case, there must be exactly one polygon on the measurement layer **M** touching or intersecting the device seed.
- There must be at least one polygon on the orientation layer **O** touching or intersecting the device seed (there can be more than one).
- The base polygon(s) **B** must be rectangular, and must have at least one coincident edge with at least one of the orientation polygons **O**. If there is more than one such coincident edge, then all those coincident edges must be parallel to each other. Those coincident edges are used to provide the orientation for the calculations.
- The measurement polygon **M** must extend beyond any **B** in the direction being measured on both sides.

For each polygon **B**, the polygon **M** is broken up into trapezoidal shapes with the trapezoids cut parallel or perpendicular to the coincident edges of **M** and **O**. For each trapezoid, three values are computed: (a,b,w). Values a and b are the average distance from **B** to the respective enclosing edge of **M**. Value w is the width of the respective trapezoid. The result of the function is an array of triplets in the form:

$a_1, b_1, w_1, a_2, b_2, w_2, \dots, a_n, b_n, w_n$

where n is the number of trapezoids. Refer to [Figure 5-4](#) and [5-5](#).

If the base polygon **B** does not satisfy the conditions required for taking valid measurements as described previously, then the value -1 is returned for all measurements:  $a_i$ ,  $b_i$ , and  $w_i$ .

When several base polygons **B** share a single measurement polygon **M**, then the trapezoidal cut lines go from one side of the **M** polygon to another across all the **B** polygons. This is true for both the case where multifinger measurements are being taken for a single device, or where separate devices are being measured. The a and b measurements for each **B** polygon are made from the **B** polygon all the way to the enclosing edges of **M**, *looking through* and ignoring any **B** polygons in between.

For the non-multifinger forms of these functions, individual property names are constructed by appending the declared property name with the letter a, b, or w, and a trapezoid number. For example, given the following Device statement definition:

```
device mp seed poly psd psd well [
    property ex
    ex = enclosure_perpendicular( seed, well, s, 3 )
]
```

the device might appear in the extracted layout netlist as follows:

```
M0 1 4 3 9 p exw1=6e-07 exa1=7e-07 exb1=8e-07 exw2=5e-07 exa2=7e-07
exb2=1e-06
```

In this example, S is the name of the source pin. Here, the device seed layer is used as **B**. Layers other than the seed can be used also.

In the multifinger forms of the functions, it is not possible to assign the function results to a PROProperty variable, so no property names are constructed.

Similarly, the ENClosure\_PARallel functions can be used to calculate a vector of properties for trapezoids that are oriented parallel to the coincident edge of **B** and **O**.

With the following property calculation:

```
device mp seed poly psd well [
    property ey
    ey = enclosure_parallel( seed, well, S, 3 )
]
```

the device might appear in the extracted layout netlist as follows:

```
M0 1 4 3 9 p eyw1=2e-07 eya1=3e-07 eyb1=4e-07 eyw2=2e-07 eya2=3e-06
eyb2=3e-07
```

## Arguments

Similar to other data retrieval functions, the first argument **B**, the **base-pin-or-layer**, may reference any pin name, device layer name or number, or auxiliary layer name or number present in the Device statement attached to the property computation program. For multifinger functions, **B** cannot be the seed layer since there can be only one seed shape per device.

The second argument **M**, the **measurement-pin-or-layer**, may reference any pin name, device layer name or number, or auxiliary layer name or number present in the Device statement.

The third argument, **O**, or **orientation-pin-or-layer**, may reference any pin or layer name present in the Device statement, as long as the respective polygon is coincident with the base polygon **B** on exactly one edge.

The fourth argument, **constraint-distance**, is a linear distance in user units. All parts of **M** farther than **constraint-distance** from a bounding box around all **B** polygons are removed from **M** for the purpose of this calculation. This argument must be a literal numeric constant with value greater than zero. Numeric expressions, local variables, and results of other data functions are not permitted.

Large values of **constraint-distance** may significantly degrade performance. For that reason, it is recommended that **constraint-distance** be as small as possible. For example, in a well-proximity calculation, suppose that enclosing WELL edges beyond six user units from the transistor gate have no significant effect on the electrical behavior of the transistor. Set **constraint-distance** equal to 6. WELL edges beyond six user units from the transistor gate will have a or b values equal to 6.

If you cannot determine the **constraint-distance** and have to use an “unlimited” distance, then specify a very large value, for example 10000. But beware of negative effects on performance. In particular, when your designs contains complex **M** polygons with many edges each.

The arguments to the enclosure functions may be declared as variables in a [Variable](#) statement in the rule file. The arguments may not be variables declared in the built-in program itself.

## Enclosure Arrays and Operators

The enclosure operations return the result in an array. Depending on the array type, you can access the values from within the Device property computation program itself, or by using the [TVF\\_numeric\\_function](#). The arrays and access methods are described as follows:

- ENClosure\_PARallel(), ENCclosure\_PERpendicular(), and ENCclosure\_VECTor()
  - [Enclosure array](#) — Methods for accessing the output from within the Device property computation program.
  - [TVF Object for Enclosure Array](#) — Using TVF interface to access the enclosure array.
- ENCclosure\_PARallel\_MULTifinger() and ENCclosure\_PERpendicular\_MULTifinger()
  - [TVF Object for Multifinger Enclosure Array](#) — Using TVF interface to access the multifinger enclosure array. This is the only access method for the multifinger array.

### Enclosure array

The results of ENCclosure\_PARallel() and ENCclosure\_PERpendicular() are *enclosure arrays*. The enclosure array is a vector of triplets. Each triplet has the form (a,b,w). Individual components of those triplets can be retrieved with the “::” operator, called the vector selection operator, from within the device computation program. For example, if S is the result of ENCclosure\_PARallel() or ENCclosure\_PERpendicular() then:

- S::a — Returns a vector of numbers containing all “a” values in S.
- S::b — Returns a vector of numbers containing all “b” values in S.
- S::w — Returns a vector of numbers containing all “w” values in S.

As shown, each of these vector selection expressions returns a value that is a simple array of numbers. These expressions can be used in functions such as [sum\(\)](#). The names a, b, and w are case-insensitive; for example, both S::a and S::A can be used.

The enclosure array does not apply to the multifinger versions of the enclosure functions. For the multifinger versions, you must use the [TVF Object for Multifinger Enclosure Array](#).

### TVF Object for Enclosure Array

You can pass the enclosure array output from a ENCclosure\_PARallel() and ENCclosure\_PERpendicular() function to a Tcl process by using [TVF\\_numeric\\_function](#). This is optional. See “[TVF Functional Interface for Device Property Calculation](#)” on page 1827 for details.

The Enclosure Array TVF Object represents the output from ENClosure\_VECTor(), ENCclosure\_PERpendicular(), and ENCclosure\_PARallel() functions. Subcommands available for an Enclosure Array Command Object are these:

- ***object slice\_count*** — returns the number of a,b,w measurement slices in the enclosure array. This function takes no arguments.
- ***object a index*** — returns the “a” measurement for a particular slice of the array. Takes the slice index as an argument.
- ***object b index*** — returns the “b” measurement for a particular slice of the array. Takes the slice index as an argument.
- ***object w index*** — returns the “w” measurement for a particular slice of the array. Takes the slice index as an argument.

### Example

The Tcl proc sum\_w function, sums the “w” measurement over the slices of an enclosure array:

```
proc sum_w { enc } {
    set w_acum 0.0
    set slice_count [ $enc slice_count ]
    # Accesses the enc command object for its slice count
    for { set i 0 } { $i<$slice_count } { incr i } {
        set w_acum [ expr { $w_acum + [ $enc w $i ] } ]
    }
    return $w_acum
}
```

The result of sum\_w operating on the enclosure array ENC is assigned to PROP\_VAL within the device property computation program with this command:

```
PROP_VAL = TVF\_numeric\_function::device_func::sum_w( ENC )
```

Note that the same result can be obtained with the following command using the [sum\(\)](#) function:

```
PROP_STR = sum( ENC::W )
```

While the Device property computation language is limited to this type of simple summation expression, the Tcl proc can perform arbitrary calculations as it iterates through the slices of the enclosure array.

## TVF Object for Multifinger Enclosure Array

The results of ENCclosure\_PARallel\_MULTifinger() and ENCclosure\_PERpendicular\_MULTifinger() are of type *multifinger enclosure array*. You must pass the multifinger enclosure array to a [TVF\\_numeric\\_function](#) for additional processing.

Subcommands available for a Multifinger Enclosure Array TVF Object are these:

- ***object finger\_count*** — returns the number of fingers in the multifinger enclosure array. This function takes no arguments.
- ***object slice\_count finger\_index*** — returns the number of a,b,w measurement slices in a particular finger from the multifinger enclosure array. Takes the finger index as an argument.
- ***object a finger\_index slice\_index*** — returns the “a” measurement for a particular finger and slice. Takes the finger and slice indices as arguments.
- ***object b finger\_index slice\_index*** — returns the “b” measurement for a particular finger and slice. Takes the finger and slice indices as arguments.
- ***object w finger\_index slice\_index*** — returns the “w” measurement for a particular finger and slice. Takes the finger and slice indices as arguments.

See the [Example with TVF Object for Enclosure Array](#), which has a similar structure.

#### Restrictions

- The layer ***B*** can be the seed layer (for non-multifinger functions) or any pin layer, pin name, or auxiliary layer.
- The layer ***M***, can be any pin layer, pin name, or auxiliary layer. The ***M*** cannot be the seed layer.
- The layer ***O***, can be any pin layer, pin name, or auxiliary layer. The ***O*** cannot be the seed layer.
- Layers ***B***, ***M***, and ***O*** must all be different layers or resolve to different layers (when pin names are used).
- The base polygon(s) ***B*** must be rectangular, and must have at least one coincident edge with the orientation polygon(s) ***O***. If there is more than one such coincident edge then all those coincident edges must be parallel to each other. The measurement polygon ***M*** must extend beyond all ***B*** polygons in the direction being measured on both sides.

---

#### Note



You can use layer operations such as [Copy](#) to circumvent some of the layer restrictions.

---

Because the functions create arrays of values, the following restrictions apply:

- The ENClosure\_PARallel() and ENCclosure\_PERpendicular() function calls may only appear in the property computation language in assignment operations or as an argument to a [TVF\\_numeric\\_function](#). The right-hand side of an assignment or TVF\_NUMerical\_FUNction() argument may contain only the ENCclosure\_PARallel() or ENCclosure\_PERpendicular() function call; it cannot be a more complex expression.

- The ENClosure\_PARallel\_MULTifinger() and ENCclosure\_PERpendiculaR\_MULTifinger() function calls may only appear in the property computation language in assignment operations to non-PROPerty variables or as an argument to a TVF\_NUMeric\_FUNction(). The right-hand side of an assignment or TVF\_NUMeric\_FUNction() argument may contain only the ENCclosure\_PARallel() or ENCclosure\_PERpendiculaR() function call; it cannot be a more complex expression.
- A property or variable receiving the result of an ENCclosure\_PARallel() or ENCclosure\_PERpendiculaR() function call may not appear as an operand in a property computation formula or expression, except in the form of a vector selection expression as described previously, and only where vector selection expressions are allowed (for example, the [sum\(\)](#) function).
- A variable receiving the result of an ENCclosure\_PARallel\_MULTifinger() or ENCclosure\_PERpendiculaR\_MULTifinger() function call can only appear as an argument to a TVF\_NUMeric\_FUNction() call.
- A property or variable receiving the result of a ENCclosure\_PARallel(), ENCclosure\_PARallel\_MULTifinger(), ENCclosure\_PERpendiculaR(), or ENCclosure\_PERpendiculaR\_MULTifinger() function call may not be assigned any other values in the Device operation.
- A property receiving the result of an ENCclosure\_PARallel() or ENCclosure\_PERpendiculaR() function call cannot be traced with the [Trace Property](#) specification statement or used in other LVS specification statements such as [LVS Filter](#), [LVS Reduce](#), [LVS Property Map](#), [LVS Split Gate Ratio](#), and so forth. Such use results in a rule file compilation error of the form:

```
Error <location and type> - this MASK mode <operation type> property
is not computed with correct type in all corresponding device
definitions: <prop>
```

- The result of an ENCclosure\_PARallel\_MULTifinger() or ENCclosure\_PERpendiculaR\_MULTifinger() function call cannot be assigned to a PROPerty. Such use results in a rule file compilation error of the form:

```
ERROR <location and type> - Multi Finger functions may not be
assigned to declared properties: <prop>.
```

## Licensing

These operations cause Calibre nmLVS to reserve a Calibre Advanced Device Properties (ADP) product license in addition to the usual licenses. Specifically, a Calibre ADP license is reserved in nmLVS if circuit extraction is performed (either flat or hierarchical) and one or more Device operations in the rule file incorporate the ENCclosure\_PARallel() or ENCclosure\_PERpendiculaR() functions. In multi-threaded operation, additional Calibre ADP licenses are reserved according to the same scheme that governs nmLVS licenses.

## Debugging

When you use the DEBUG statement with these functions for a particular device, measured values and error diagnostics are sent to the transcript.

## enclosure\_vector

### **ENClosure\_VECtor (*measurement-pin-or-layer*, *constraint-distance*)**

This function is used for well proximity and stress effect calculations. It has fewer capabilities than the [device::enclosure\\_measurements](#) or the [ENClosure\\_PARallel](#) and [ENClosure\\_PERpendicular](#) Functions, and is much less efficient. However, it is simpler to use and it does not require the Calibre Advanced Device Properties (ADP) product license.

#### Description

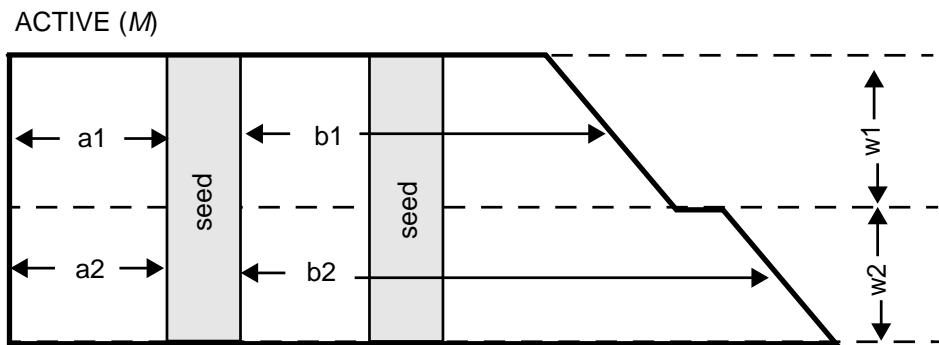
The function measures a set of values regarding the enclosure of the specified ***measurement-pin-or-layer*** over the seed shape of the device. The ***measurement-pin-or-layer*** is henceforth referred to as ***M***. In order to take valid measurements, the seed shape of the device must be rectangular, must be completely enclosed within ***M***, and must coincide with ***M*** on exactly two opposite edges that must be parallel to each other (for example, the top and bottom edges in [Figure 5-6](#)).

***M*** is then broken up into trapezoidal shapes with the trapezoids cut parallel to the coincident edges of ***M*** and the seed shape. For each trapezoid, three values are computed: (a, b, and w). Values a and b are the average distance from the seed shape to the respective enclosing edge of ***M***. Value w is the width of the respective trapezoid. The result of the function is an array of triplets in this form:

$$a_1, b_1, w_1, a_2, b_2, w_2, \dots, a_n, b_n, w_n$$

where n is the number of trapezoids. Note that the measurements a, b, and w are not always oriented in the same way versus the device seed shape (left-to-right or bottom-to-top, and so forth). However, the tool will calculate the vectors containing these measurements consistently.

**Figure 5-6. ENCclosure\_VECtor() Vector Elements**



(Measurements shown in [Figure 5-6](#) are for the seed on the left. The other seed would have its own set of measurements, which are not shown. The dotted line indicates a trapezoidal cutline.)

If the seed shape of the device does not satisfy the conditions required for taking valid measurements as described above, then the value -1 is returned for all measurements:  $a_i$ ,  $b_i$ , and  $w_i$ .

When several seed shapes share a single  $M$  polygon, then the trapezoidal cut lines go from one side of the  $M$  polygon to another, across all the seed shapes. The a and b measurements for each seed shape are made from the seed shape all the way to the enclosing edges of  $M$ , “looking through” and ignoring any seed shapes in between. This is shown in the diagram.

Individual property names are constructed by appending the declared property name with the letter a, b, or w and a trapezoid number. For example, given the following Device definition:

```
device mp seed poly psd psd nwell <active> [
    property s
    s = enclosure_vector ( active, 3 )
]
```

the device might appear in the extracted layout netlist as follows:

```
M0 1 4 3 9 N sw1=6e-07 sa1=7e-07 sb1=8e-07 sw2=5e-07 sa2=7e-07 sb2=1e-06
```

## Arguments

The argument  $M$  (*measurement-pin-or-layer*) may reference any pin name, layer name/number, or auxiliary layer name/number present in the Device statement to which the property computation program is attached.

The second argument, *constraint-distance*, is a linear distance in user units. All parts of  $M$  farther than *constraint-distance* from the seed shape are removed from  $M$  for the purpose of this calculation. This argument must be a literal numeric constant with value greater than zero. Numeric expressions, local variables, and results of other data functions are not permitted.

Large values of *constraint-distance* may significantly degrade performance. For that reason, it is recommended that *constraint-distance* be as small as possible.

For example, for a stress-effect calculation, suppose that enclosing ACTIVE edges beyond five user units from the transistor gate has no significant effect on the electrical behavior of the transistor. Set *constraint-distance* to 5. ACTIVE edges beyond five user units from the transistor gate will have a or b values equal to 5.

If you cannot determine the *constraint-distance* and have to use an “unlimited” distance, then specify a very large value, for example 10000. But beware of negative effects on performance, in particular when your designs may contain complex  $M$  polygons with many edges each.

The arguments to the ENClosure\_VECtor() function may be declared as variables in a [Variable](#) statement in the rule file. The arguments may not be variables declared in the built-in program itself.

## Vector Form and Operators

The result of ENCclosure\_VECtor() is of type *vector of triplets*. Each triplet has the form (a, b, w). Individual components of those triplets can be retrieved with the “::” operator, called the *vector selection* operator. For example, if S is the result of ENCclosure\_VECtor then:

S::a — Returns a vector of numbers consisting of all “a” values in S.

S::b — Returns a vector of numbers consisting of all “b” values in S.

S::w — Returns a vector of numbers consisting of all “w” values in S.

As shown, each of these vector selection expressions returns a value that is a simple array of numbers. These expressions can be used in functions such as SUM(), described under “[TVF Functional Interface for Device Property Calculation](#)” on page 1827. The names a, b, and w are case-insensitive; for example, both S::a and S::A can be used.

See “[TVF Object for Enclosure Array](#)” on page 1811 for a description of how to use the output of the ENClosure\_VECtor() function in a TVF program.

### Restrictions

Because the function creates a vector of values, several restrictions apply:

- The ENCclosure\_VECtor() function call may appear in the property computation language in assignment operations only. The right-hand side of the assignment may contain only the ENCclosure\_VECtor() function call; it cannot be a more complex expression.
- A property or variable receiving the result of an ENCclosure\_VECtor() function call may not appear as an operand in a property computation formula or expression, except in the form of a vector selection expression as described previously, and only where vector selection expressions are allowed (for example, the SUM() function).
- A property or variable receiving the result of an ENCclosure\_VECtor() function call may not be assigned any other values in the Device operation.
- A property receiving the result of an ENCclosure\_VECtor() function call cannot be traced with the [Trace Property](#) specification statement or used in other LVS specification statements such as [LVS Filter](#), [LVS Reduce](#), [LVS Property Map](#), [LVS Split Gate Ratio](#), and similar statements. Such use results in a rule file compilation error of the form:

```
Error <location and type> - this MASK mode <operation type> property  
is not computed with correct type in all corresponding device  
definitions: <property>
```

- As stated previously, the seed shape of the device must be rectangular, and it must coincide with the **M** object on exactly two opposite edges.

## instance

### INSTance()

#### Description

Returns the instance number of the current device instance. Each device extracted has a non-negative instance number that is unique over all devices of all types extracted during that extraction run. LVS reports the instance numbers of devices as part of the property discrepancy report. This function allows conversion of these numbers into a property value for use in other contexts.

Instance numbers from this function are not valid if you use [LVS Push Devices YES](#).

## named\_net

**NAMED\_NET("net-name")**

### Description

Returns the number of a net. It requires a net name enclosed in quotation marks ("") as an argument and returns the number of the net that has that name.

If there is no net with that name, it returns the special value 0, which indicates that there is no net with the given name. This function is not implemented hierarchically.

## Perimeter Functions

The six perimeter functions calculate the perimeter length or portion of a perimeter with varying restrictions. See [Figure 5-7](#) and [Table 5-5](#) for a summary of how the perimeter functions operate.

See the section [“Pin and Layer References”](#) on page 1786 for details about the *pin-or-layer* argument.

### perimeter

**PERIMeter(*pin-or-layer*)**

### Description

Returns the total length in meters of the perimeter of the shapes in the specified pin or on the specified layer.

### perimeter\_inside

**PERIMeter\_INside(*pin-or-layer*, *pin-or-layer*)**

### Description

Returns the total length in meters of the parts of perimeters on the first pin or layer that lie strictly inside shapes of the second pin or layer. Corresponds to portion 1 in Figure [5-7](#).

### perimeter\_outside

**PERIMeter\_OUTside(*pin-or-layer*, *pin-or-layer*)**

### Description

Returns the total length of the parts of perimeters on the first pin or layer that lie strictly outside shapes of the second pin or layer. Corresponds to portion 3 in Figure [5-7](#).

## perimeter\_coincide

**PERIMeter\_COincide(*pin-or-layer*, *pin-or-layer*)**

### Description

Returns the total length in meters of the parts of perimeters on the first pin or layer which coincide with the perimeter of the second pin or layer. Corresponds to portions 2a and 2b in Figure 5-7.

## perimeter\_coincide\_inside

**PERIMeter\_COincide\_INside(*pin-or-layer*, *pin-or-layer*)**

### Description

Returns the total length in meters of the parts of perimeters on the first pin or layer that coincide with parts of the perimeters of shapes of the second pin or layer and where the shape of the first layer is inside the shape of the second layer. Corresponds to portion 2a in Figure 5-7.

## perimeter\_coincide\_outside

**PERIMeter\_COincide\_OUTside(*pin-or-layer*, *pin-or-layer*)**

### Description

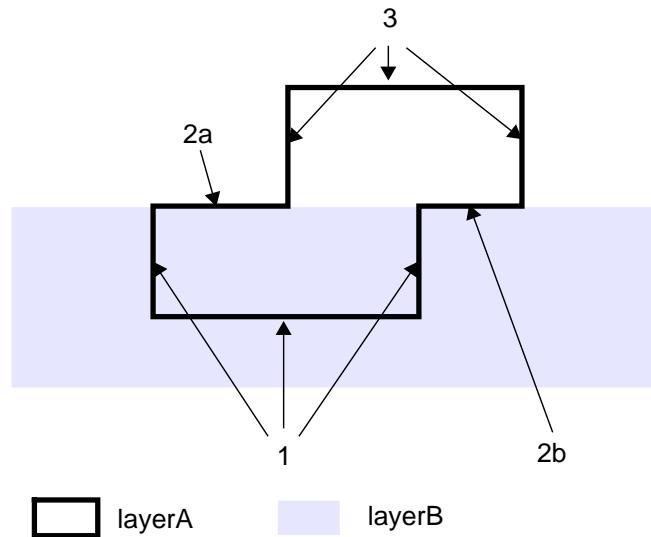
Returns the total length in meters of the parts of perimeters on the first pin or layer that coincide with parts of the perimeters of shapes of the second pin or layer and where the shape of the first layer is outside the shape of the second layer. Corresponds to portion 2b in Figure 5-7.

### Summary of Perimeter Functions

Table 5-5 describes the six perimeter functions that deal with perimeter length. The function PERIMeter(*pin\_or\_layer*) gives a simple perimeter while the other five functions report the length of portions of perimeters on one pin or layer as they relate to shapes on a second pin or layer. All values are assumed to be in meters by default.

Consider two overlapping shapes on layers A and B as shown in Figure 5-7.

**Figure 5-7. Perimeter Relationships**



The perimeter of layer A can be divided into three portions depending on its relation to the shape on layer B. These three portions have been labeled with numbers as follows:

- 1: PERIMETER of A is strictly INSIDE the shape of B.
- 2: PERIMETER of A and of B COINCIDE.
- 3: PERIMETER of A is strictly OUTSIDE the shape of B.

The portion labeled 2 can be further subdivided into two portions depending on the relation of the shapes where the perimeters are coincident. These have been labeled as follows:

- 2a: PERIMETER of A and of B COINCIDE, and shape A lies INSIDE shape B.
- 2b: PERIMETER of A and of B COINCIDE and shape A lies OUTSIDE shape B.

Use of the three keywords INside, COincide, and OUTside as illustrated above gives rise to the following six perimeter functions:

**Table 5-5. Perimeter Functions**

Function Name	Portions Measured in <a href="#">Figure 5-7</a>			
PERIMeter	1	2a	2b	3
PERIMeter_INside	1			
PERIMeter_COincide		2a	2b	
PERIMeter_OUTside				3
PERIMeter_COincide_INside		2a		
PERIMeter_COincide_OUTside			2b	

## pin\_net

**PIN\_NET("pin-name")**

### Description

Returns the number of a net. Requires a pin name as its argument and returns the number of the net attached to the pin.

The following example uses case 1 if the D pin is connected to VDD and case 2 otherwise:

```
if ( pin_net(D) == named_net( "VDD" ) )
    // case 1
else
    // case 2
```

In hierarchical applications, this function returns node number local to the cell, which is a limitation.

## precision

**PRECISION()**

### Description

Returns the current value of the process precision (1000 by default). This function takes no arguments. The precision is the number of database units per user-defined unit.

## sum

**SUM (vector-expression)**

### Description

SUM() applies the specified arithmetic expression to each element of the operand array or arrays and returns the sum of the results.

The SUM() function can be used to sum elements of the vectors that are output by the Enclosure functions (see “[ENClosure\\_PARallel and ENCclosure\\_PERpendicular Functions](#)” and “[enclosure\\_vector](#)”). This function can be used in conjunction with the Enclosure functions,

### Arguments

- **vector-expression** — A regular arithmetic expression that must contain one or more vector selection expressions, each returning a simple array of numbers. In addition, **vector-expression** may contain numeric constants, arithmetic operators, numeric functions, and so forth.

### Returns

SUM() returns a floating point number.

For example, if S is the result of an enclosure function call, then the expression

`SUM(S::w)`

returns the sum of all individual “w” values from S. Similarly, the expression

```
SUM(S::a - 3 * S::b + 7)
```

returns the sum of all  $(a_i - 3 * b_i + 7)$  values from S, where i is a running index over S.

### Restrictions

Several restrictions apply to the ***vector-expression*** used as an argument to SUM():

1. The ***vector-expression*** must contain at least one vector selection expression.
2. If ***vector-expression*** contains multiple vector selection expressions, they must all be from the same vector-typed variable.
3. The ***vector-expression*** must not contain another SUM() function call (although it may contain a property or local variable that is generated by another SUM() function).

### Example 1

```
device mp seed poly psd nwell <active> [
    property w
    s = enclosure_perpendicular( seed, active, psd, 3 )
    w = sum( s::w )
]
```

The property w contains the total width of the seed shape across the active layer.

### Example 2

```
device mp seed poly psd nwell <active> [
    property w, len, saeff, sbeff
    w = perim_co( psd, seed ) / 2
    len = area( seed ) / w
    s = enclosure_perpendicular( seed, active, psd, 3 )
    saeff = w / sum( s::w / ( s::a + 0.5 * len ) ) - 0.5 * len
    sbeff = w / sum( s::w / ( s::b + 0.5 * len ) ) - 0.5 * len
]
```

The property saeff is calculated by summing the results of the expression:

$s::w / (s::a + 0.5 * len)$

for each corresponding value of w and a in the results vector s, then performing the remaining calculation:

$saeff = w / <\text{result\_of\_sum}> - 0.5 * len$

Property sbeff is computed in a similar way.

## text\_numeric

**TEXT\_NUMerIC (layer, numeric-default)**

### Description

This function calculates the numeric values of text objects in the layout that intersect the device seed shape.

The device recognizer looks for text objects on *layer* such that the text location intersects the seed polygon of the device. If a text object is found, then its numeric value is returned by the TEXT\_NUMERIC() function. If more than one text object on *layer* intersects the seed polygon, then the device is rejected and classified as BAD.

The ***numeric-default*** is returned if a numeric value can not be determined, as explained in the section “Returns”.

### Arguments

- ***layer*** — Must be the name or number of an original layer or layer set, which is specified in a TEXT PROPERTY LAYER parameter in a **Device** operation.
- ***numeric-default*** — Must be a non-negative, floating-point constant.

### Returns

When successful, the text\_numeric function returns a floating-point numeric values that may be assigned to numeric-valued variables or properties. The conversion process is described in the section “[Conversion From Text to Numeric Floating-Point Value](#)”.

[Table 5-6](#) lists the possible return values of text\_string.

**Table 5-6. Return Values of TEXT\_NUMERIC Function**

Outcome	Return Value
Found one text object on <i>layer</i> that intersects the seed polygon.	Floating point value.
More than one text object on <i>layer</i> intersects the seed polygon.	The device is rejected and classified as BAD
No text objects on <i>layer</i> intersect the seed polygon.	The <b><i>numeric-default</i></b> value.
A text object exists on <i>layer</i> intersecting the seed polygon of the device, but the text object is empty or zero-length.	The <b><i>numeric-default</i></b> value.
A text object exists on <i>layer</i> intersecting the seed polygon of the device, but it cannot be evaluated as a valid, floating-point, numeric value.	The <b><i>numeric-default</i></b> value.

### Example

Here is a complete example in the context of a Device operation:

```
DEVICE MP Pseed poly(G) PSD(S) PSD(D)
TEXT PROPERTY LAYER tpl1 tpl2
[
    PROPERTY NumProp1, NumProp2, AreaSeed
    NumProp1 = text_num( tpl1, 0.0 )
    NumProp2 = text_num( tpl2, 1.1 )
    AreaSeed = area( Pseed )
]
```

Each device instance recognized by the above device operation has a property named AreaSeed with a numeric value representing the area of the device seed. Each device instance also has a numeric-valued property named NumProp1 with either the value 0.0 in cases where no valid numerically-valued text objects on layer tpl1 intersect the device seed, or the numeric value of a text object on layer tpl1. Similarly, each device instance also has a numeric-valued property named NumProp2 with either the value 1.1 or the numeric value of a text object on layer tpl2, which intersects the seed.

### Conversion From Text to Numeric Floating-Point Value

This section is adapted from the UNIX manual page strtod(3D).

When a text exists on *layer* intersecting the seed polygon of the device, the TEXT\_NUMERIC() function converts the initial portion of the text to a numeric floating-point value as follows:

First it decomposes the text into three parts:

- an initial, possibly empty, sequence of whitespace characters
- a *subject sequence* interpreted as a floating-point value
- a final sequence of one or more unrecognized characters

Then it attempts to convert the subject sequence into a floating-point value, and returns the result.

The expected form of the subject sequence is an optional + or - sign, then a non-empty sequence of digits optionally containing the radix character (which is restricted to the period character ‘.’), then an optional exponent part. An exponent part consists of the character ‘e’ or ‘E’, followed by an optional sign, followed by one or more decimal digits. The *subject sequence* is defined as the longest initial subsequence of the input text, starting with the first non-whitespace character, that is of the expected form. The subject sequence is empty if the text is empty or consists entirely of whitespace characters, or if the first character that is not whitespace is other than a sign, a digit, or the radix character.

If the subject sequence has the expected form, then the sequence starting with the first digit or the radix character (whichever occurs first) is interpreted as a numeric, floating point value, except that if neither an exponent part nor a radix character appears, then a radix character is assumed to follow the last digit in the sequence of characters. If the subject sequence begins with a minus sign (-), the value resulting from the conversion is negative.

## text\_string

**TEXT\_STRING** (*layer*, “*string-default*”)

### Description

This function returns string values of text objects in the layout that intersect the device seed shape.

The device recognizer looks for text objects on *layer* such that the text location intersects the seed polygon of the device. If a text object is found, then its value is returned by the

TEXT\_STRING() function as a character string. If more than one text object on *layer* intersects the seed polygon, then the device is rejected and classified as BAD.

If no text objects on *layer* intersect the seed polygon, then TEXT\_STRING() returns the *string-default* value.

#### Arguments

- *layer* — Must be the name or number of an original layer or layer set, which is specified in a TEXT PROPERTY LAYER parameter in a [Device](#) operation.
- *string-default* — A required string value enclosed in quotes.

#### Returns

[Table 5-7](#) lists the possible return values of text\_string.

**Table 5-7. Return Values of TEXT\_STRING() Function**

Outcome	Return Value
Found one text object on <i>layer</i> that intersects the seed polygon.	String value of text object
More than one text object on <i>layer</i> intersects the seed polygon.	The device is rejected and classified as BAD
No text objects on <i>layer</i> intersect the seed polygon.	The <i>string-default</i> value.

#### Example

Here is a complete example in the context of a Device operation:

```
DEVICE MP Pseed poly(G) PSD(S) PSD(D)
    TEXT PROPERTY LAYER tpl1 tpl2
    [
        PROPERTY StrProp1, StrProp2, AreaSeed
        StrProp1 = text_str( tpl1, "default1" )
        StrProp2 = text_str( tpl2, "default2" )
        AreaSeed = area( Pseed )
    ]
```

Each device instance recognized by the this Device operation has a property named AreaSeed, with numeric value representing the area of the device seed. Each device instance also has a string-valued property named StrProp1, with either the value default1 in cases where no text objects on layer tpl1 intersect the device seed, or the string value of a text object on layer tpl1 in cases where exactly one text object on layer tpl1 intersects the device seed. Similarly, each device instance also has a string-valued property named StrProp2, with either the value default2, or the string value of a text object on layer tpl2.

## TVF Functions for Device Property Computation

The Device property calculation language supports calls to Tcl procedures defined in TVF FUNCTION statements in the SVRF rule file (see “[TVF Function](#)” on page 1906 for details

about this statement). This is used only in Calibre and requires a Calibre Advanced Device Properties (ADP) product license in addition to the usual licenses.

See “[TVF Functional Interface for Device Property Calculation](#)” on page 1827 for a full description of how to use these functions for device property computation and a description of the Tcl command objects used for the different types of enclosure arrays.

## TVF\_numeric\_function

**TVF\_NUMERIC\_FUNCTION::tvf\_func::tcl\_proc( parameter [, ... ] )**

### Description

Calls a Tcl procedure (*tcl\_proc*) that returns a numeric value. The procedure *tcl\_proc* must be defined within the TVF Function *tvf\_func*. The *parameter* is passed to the *tcl\_proc*.

## TVF\_string\_function

**TVF\_STRING\_FUNCTION::tvf\_func::tcl\_proc( parameter [, ... ] )**

### Description

Calls a Tcl procedure (*tcl\_proc*) that returns a string value. The procedure *tcl\_proc* must be defined within the TVF Function *tvf\_func*. The *parameter* is passed to the *tcl\_proc*.

## unit\_capacitance

**UNIT\_CAPacitance()**

### Description

Returns the current value of the unit capacitance as specified in the rule file by the [Unit Capacitance](#) statement. This function takes no arguments. The unit-capacitance is the magnitude of the user capacitance unit expressed in farads.

## unit\_length

**UNIT\_LENGTH()**

### Description

Returns the current value of the process [Unit Length](#), normally 1E-6. This function takes no arguments. The unit length is the magnitude of a user length unit expressed in meters.

## unit\_resistance

**UNIT\_REStance()**

### Description

Returns the current value of the unit resistance as specified in the rule file by the [Unit Resistance](#) statement. This function takes no arguments. The unit resistance is the magnitude of the user resistance unit expressed in ohms.

## x\_location

### X\_LOCATION(*pin-or-layer*)

#### Description

Returns the x coordinate in user units associated with the given pin or layer.

For seed layers, the function returns the x coordinate of the hierarchically lowest, leftmost point of the seed shape.

For pin layers and auxiliary layers, the function returns the x coordinate of an arbitrarily chosen point on the hierarchically lowest, leftmost of the edges common to the given pin or layer and to the seed shape.

## y\_location

### Y\_LOCATION(*pin-or-layer*)

#### Description

Returns the y coordinate in user units associated with the given pin or layer.

For seed layers, the function returns the y coordinate of the hierarchically lowest, leftmost of the point of the seed shape.

For pin layers and auxiliary layers, the function returns the y coordinate of an arbitrarily chosen point on the hierarchically lowest, leftmost of the edges common to the given pin or layer and to the seed shape.

## TVF Functional Interface for Device Property Calculation

The Device property calculation language supports calls to Tcl procedures defined in TVF Function statements in the SVRF rule file (see “[TVF Function](#)” on page 1906 for details about this statement). The TVF Function statement is used only in Calibre. The Device TVF interface requires a Calibre Advanced Device Properties (ADP) product license in addition to the usual licenses.

The Tcl procedure may be called through two Device property calculation functions called TVF\_NUMeric\_FUNction() and TVF\_STRING\_FUNction(). The syntaxes are as follows:

**TVF\_NUMeric\_FUNction::tvf\_func::tcl\_proc( parameter [, ... ] )**

**TVF\_STRING\_FUNction::tvf\_func::tcl\_proc( parameter [, ... ] )**

where these are the definitions of the elements:

***tvf\_func*** — a TVF Function statement defined in the rule file. It contains one or more Tcl procs.

***tcl\_proc*** — a Tcl proc name. A Tcl proc is a procedure within a TVF Function statement.

**parameter** — parameter passed from the Device script to the Tcl script. A list of parameters is delimited by commas.

The difference between these functions is TVF\_NUMeric\_FUNction() returns a numeric value, and TVF\_STRING\_FUNction() returns a string value.

Parameters passed to the Tcl procedure can consist of any of the following types:

- A numeric argument (a number, result of a numeric function, or a numeric variable).
- A string argument (a result of a string function or a string variable).
- An enclosure array, which is the output of a [enclosure\\_vector](#), [enclosure\\_parallel](#), or [enclosure\\_perpendicular](#) function. See “[TVF Object for Enclosure Array](#)” on page 1811.
- A multifinger enclosure array, which is the output of a [enclosure\\_parallel\\_multifinger](#) or [enclosure\\_perpendicular\\_multifinger](#) function. See “[TVF Object for Multifinger Enclosure Array](#)” on page 1812.
- An array or vector which is the output of the device dfm functions:
  - “[TVF Object for dfm\\_numeric\\_array\(\)](#)” on page 1790
  - “[TVF Object for dfm\\_vector\\_value\(\)](#)” on page 1792
  - “[TVF Object for dfm\\_vector\\_array\(\)](#)” on page 1796

The Tcl proc called by TVF\_NUMeric\_FUNction() must return a single floating-point number that is, in turn, assigned to a property or variable to be used in the remainder of Device program.

Similarly, the Tcl proc called by TVF\_STRING\_FUNction() must return a string that is, in turn, assigned to a string property or variable to be used in the remainder of Device program.

The arguments passed in to a TVF function Tcl proc are visible from within the proc as TVF objects. The interface available for the TVF object depends upon the type of the TVF object passed in. Each of the TVF objects are described with the command that produces them.

In addition, these device TVF commands can be used to convert DFM-style rectilinear measurement vectors into Device enclosure-vector-style width and length measurements:

- “[device::dfm\\_vec\\_measurements](#)” on page 1797
- “[device::scaled\\_dfm\\_vec\\_measurements](#)” on page 1806

## Example Device Property Computation with TVF Function

Consider the following device property computation program:

```
DEVICE MN ngate ngate sd sd <diff> TEXT PROPERTY LAYER dtext
[
  PROPERTY W, L, PROP_A, PROP_STR
  ENC = enclosure\_perpendicular( ngate, diff, s, 100 )

  // Demonstration of calculations using Device TVF Functions
  W = TVF_numeric_function::device_func::sum_w( ENC )
  L = area( ngate ) / W
```

```

StrVar = text_string( dtext, "" )
PROP_A = TVF_numeric_function::device_func::get_prop( StrVar, "a", W, L
)
PROP_STR = TVF_string_function::device_func::calc_str( ENC, W, L )
]

```

Three Tcl procs are called from within this program. They are all from the TVF FUNCTION statement named device\_func.

The first proc called is sum\_w. The variable ENC, representing the output enclosure array from the ENClosure\_PERpendicula function, is passed in as an argument.

Similarly, the second function, get\_prop, is passed a string variable StrVar, the string “a”, and the numeric property values W and L. This Tcl proc makes use of the numeric and string objects as described in [“Numeric Command Object”](#) on page 1830 and [“String Objects in TVF\\_NUMeric\\_FUNction Procedures”](#) on page 1831.

The third function, calc\_str is passed the ENC variable along with the numeric property variables W and L. The return values of W, PROP\_A, and PROP\_STR are all set by the values of the Tcl procs that are called.

Here is the full text of the TVF FUNCTION used for the device computation program:

```

TVF FUNCTION device_func /*

proc sum_w { enc } {
    set w_acum 0.0
    set slice_count [ $enc slice_count ]
    for { set i 0 } { $i<$slice_count } { incr i } {
        set w_acum [ expr { $w_acum + [ $enc w $i ] } ]
    }
    return $w_acum
}

proc get_prop { text_input key_input W L } {
    set rv 0
    set text_string [ $text_input string ]
    set width [ $W ]
    set key [ $key_input string ]
    if {$text_string != "empty"} {
        regexp "$key=(\S+)" $text_string junk rv
    }
    return [ expr { $rv / $width } ]
}

proc calc_str { enc W L } {
    set acum ""

    set slice_count [ $enc slice_count ]
    set acum "$slice_count"
    for { set i 0 } { $i<$slice_count } { incr i } {
        set acum "$acum:[ $enc w $i ]:[ $enc b $i ]"
    }
    return $acum
}
*/

```

## Numeric Command Object

The numeric object represents a simple number, numeric variable, or output from a numeric device property function. The object can be evaluated using [ ] to access the value contained in the object. For example, to access the value for L passed in to the calc\_eff\_a function, evaluate the command argument variable \$L as follows:

```
proc calc_eff_a { enc W L } {  
    ...  
    set length [$L]  
    // Accesses L command object for its numeric value  
    ...  
}
```

## String Arguments in TVF\_NUMERIC\_FUNction

[TVF\\_numeric\\_function](#) accepts strings and string properties as arguments. These arguments are available inside the Tcl proc as TVF string objects. Strings may be passed as quoted text, values derived from the Device statement's [text\\_string\(\)](#) function, or [Variable](#) statements from the rule file:

```
VARIABLE vProp "p2=4.3"  
  
DEVICE MN(nfet) ngate ngate(g) nsd(s) nsd(d) psub(b)  
TEXT PROPERTY LAYER DTEXT [  
    PROPERTY p1, p2, p3  
    StrProp? = text\_string( DTEXT, "" )  
    p1 = tvf_num_fun::device_func::scan_text( StrProp? )  
    p2 = tvf_num_fun::device_func::scan_text( vProp )  
    p3 = tvf_num_fun::device_func::scan_text( "p3=99" )  
]
```

In this example, the Device statement searches the layer DTEXT for text objects to be used for assigning property values. Within the property computation section, three properties are declared: p1, p2, and p3. The TEXT\_STRING function assigns the values of text objects on layer DTEXT that intersect ngate polygons to the StrProp? variable. If no objects are found on an ngate polygon, then nothing is assigned to StrProp? for that polygon.

The three TVF\_NUMERIC\_FUNction() calls in this example all call a proc named scan\_text. In the first case, the value of the variable StrProp? is passed as an argument to scan\_text. In the second case, the value of the rule file variable vProp is passed to scan\_text. In the third case, the string “p3=99” is passed to scan\_text.

The scan\_text proc would return a string for each of the three cases in the example. The properties p1, p2, and p3 would each be assigned string values accordingly.

## String Objects in TVF\_NUMERIC\_FUNction Procedures

Strings passed as arguments to [TVF\\_numeric\\_function](#) are accessed inside the Tcl proc as Tcl string objects. These Tcl objects accept the single function called **string** to return a Tcl string representing the object that was passed in. The syntax is this:

***object* string** — returns a Tcl string representing the string object.

For example, strings passed to [TVF\\_NUMERIC\\_FUNction\(\)](#) as follows:

```
p1 = tvf_num_fun::device_func::scan_string( "p1=3e-06 p2=2.3", "p1" )
```

can be accessed as follows:

```
TVF FUNCTION device_func /*

# this proc expects key-value pairs linked with "=" and that the pairs are
# delimited by spaces. the proc also expects a separate key value.
# scan_string ("p1=3e-06 p2=2.3", "p1") returns "3e-06"

proc scan_string {text_input key_input} {
    set rv 0
    set text_string [ $text_input string ] # get the key-value pairs
    set key [ $key_input string ]          # get the key
    if {$text_string != "empty"} {
        regexp "$key=(\\S+)" $text_string junk rv
        # performs a regular expression match for the key-value pair in
        # $text_string. the entire string match is sent to junk. the value
        # from (\\S+) is sent to the variable rv.
    }
    return $rv
}
*/ ]
```

## Device TVF Programming Considerations

There are several topics to consider when you program in TVF, as given in these sections:

Performance Notes .....	1831
Multithreading.....	1832
Debugging.....	1832
Numeric Underflow and Overflow in Tcl.....	1832
Backward-Compatible Syntax .....	1833

### Performance Notes

When writing Tcl procs for use in the device calculation module, remember that DEvice built-in language functions are called and evaluated many times during a device extraction run. Be sure to use standard Tcl performance coding practices such as the use of {} in expr statements. Also be aware that operations like list lookup, list sorting, and management of complex data structures may take significant time during device extraction.

## Multithreading

In multi-threaded runs, only one Tcl interpreter is created and all access to the interpreter is sequential. This means TVF FUNCTION blocks are processed one-at-a-time. However, Device recognition is still performed in multi-threaded mode by default in an MT/MTflex run

---

### Note



Using LVS Show Seed Promotions YES causes device recognition to run single-threaded. It is usually best to avoid using this statement in a production environment.

---

## Debugging

When the rule file is compiled, all TVF FUNCTION blocks referenced by Device statements are run with dummy values. If errors occur during this run of the proc (when no device data is present) the rule file will fail to compile. Arithmetic errors like divide by 0 are ignored during this process.

If errors occur in the Tcl proc during actual device recognition, the value of 0 is assigned to the property being calculated. Errors issued by Tcl display error information provided by Tcl in the Calibre transcript. To see the text of the error being generated, include a DEBUG statement in the device property calculation section. For example, the following statement produces debug information including Tcl runtime errors for devices 0-99 for any cell that has devices.

```
DEVICE MN NGATE NGATE SD SD <DIFF>
[
    DEBUG 0-99
    PROPERTY W, L, SEFFA, SEFFB
    ENC = ENC_PER( NGATE, DIFF, s, 100 )
    // Calculations using TVF Functions
    W = TVF_NUM_FUN::device_func::sum_w( ENC )
    L = AREA( NGATE ) / W
    SEFFA = TVF_NUM_FUN::device_func::calc_eff_a( ENC, W, L )
    SEFFB = TVF_NUM_FUN::device_func::calc_eff_b( ENC, W, L )
]
```

Use of the Tcl “puts” function can also be useful in debugging errors during execution.

## Numeric Underflow and Overflow in Tcl

Numeric underflow and overflow in Tcl expressions cause errors to be issued, resulting in a 0 return value to the Device property computation script. Calibre explicitly overrides the exp() and pow() functions in Tcl to avoid this issue since these functions are common reasons for underflow errors occurring in numeric calculations.

In Tcl 8.5, this behavior will change so that all underflow values are treated as 0 and overflow values are treated as the floating-point symbolic value Inf (the infinite value). In the mean time, calculations that might involve underflow not related to pow() and exp(), and any calculation that might involve overflow should be protected by the Tcl “catch” command.

## Backward-Compatible Syntax

The original form of TVF\_NUMeric\_FUNction() is as follows:

```
TVF_NUMeric_FUNction( "proc_name", "device_function", parameter [, parameter ...] )
```

For example:

```
PROP_A = TVF_NUM_FUN( "get_prop", "device_func", ENC, StrVal, W, L )
```

This is identical to the current syntax:

```
PROP_A = TVF_NUM_FUN::device_func::get_prop( ENC, StrVal, W, L )
```

## Well Proximity Calculation Methods and Examples

Well proximity calculation methods have undergone an evolution in Calibre. This section details the methods that have been used over many years. Currently, the recommended function for this application is [device::enclosure\\_measurements](#). This function should be used instead of others documented in this section. This is for performance. The older methods are presented for historical reasons.

The enclosure functions, the [DFM Property](#) data retrieval functions, and the Device TVF functions, when used together, are useful for MOS device *well proximity* and shallow trench isolation (STI) *stress effect* calculations and similar applications. They are primarily used to extract physical parameters for the newer BSIM4 models.

## Comparison of Enclosure Calculation Functions and Methods

The enclosure functions consist of the following:

- [device::enclosure\\_measurements](#) function — Used only in Calibre and requires a Calibre Advanced Device Properties (ADP) product license. This function is similar to the ENClosure\_\* functions, but has better performance in most cases. This is the recommended function for well enclosure calculations.

See the command description for examples using [device::enclosure\\_measurements](#).

- [DFM Property Data Retrieval Functions](#) — Used only in Calibre and require a Calibre Advanced Device Properties (ADP) product license.

For information about using the DFM Property operation for device enclosure calculations, see [“Performing Well Enclosure or Stress Effect Calculations with DFM Property”](#) on page 1837.

- [ENClosure\\_PARallel](#) and [ENClosure\\_PERpendicular](#) Functions — Used only in Calibre and requires a Calibre Advanced Device Properties (ADP) product license. These are older functions that do not perform as well as [device::enclosure\\_measurements](#) for well proximity applications.

See [“Well Proximity Property Computation Examples”](#) on page 1834 for examples. The examples include computation with and without the TVF functional interface. The TVF

functional interface gives better performance. See the command reference pages for full details on the commands.

- [enclosure\\_vector](#) function— Used only in Calibre and requires a Calibre nmLVS product license. For well proximity applications, this operation has poorer performance than those listed previously.

**Table 5-8** outlines differences between how the CalibreDFM\_DEVICE package and the enclosure\_parallel and enclosure\_perpendicular functions calculate enclosure measurements.

**Table 5-8. Differences in Enclosure Calculations**

Measurement	ENC_PAR, ENC_PER Method	CalibreDFM_DEVICE Method
Merging of source/drain measurements.	Combines two s/d regions into <width a b> segments.	Reports each s/d region separately in <width value> segments.
Angled gates and gates with bends.	Returns a vector populated with -1.	Measures angled gates and gates with bends.
Non-Manhattan measurement/base intersections.	Measures using the normal rules for angled measurement layer. See “ <a href="#">ENClosure_PARallel and ENCclosure_Perpendicular Functions</a> ” on page 1807.	Takes no measurement and populates the vector with the maximum search distance.
Orthogonal (with respect to database axes) gates with holes.	Returns a vector populated with -1.	Measures normally, ignoring the holes.
Angled well.	Reports enclosure at midpoint of non-Manhattan edge.	Reports enclosure at closest endpoint of non-Manhattan edge.
Missing well.	Reports 0.0 enclosure when no well is present for a segment.	Reports maximum search distance when no well is present.

## Well Proximity Property Computation Examples

The examples in this section use the [ENClosure\\_PARallel](#) and [ENCclosure\\_Perpendicular Functions](#). These are presented mostly for historical reasons. See [device::enclosure\\_measurements](#) for a more efficient function.

### Example of Weighted Average Enclosure Calculation

In this example, pxh is a weighted average of nwell enclosure over the transistor seed in the “horizontal” direction, and pxv is a weighted average of nwell enclosure over the transistor seed

in the “vertical” direction. Here, horizontal is the direction from source and drain, and vertical is the direction of the poly wire.

Weighting is by the distance ( $w_i$ ) over which each particular enclosure measurement applies, relative to the total dimension of the transistor seed (w or l, respectively).

```
device mp seed poly psd psd nwell [
    property 2w, 2l, pxh, pxv
    2w = perimeter_coincide( psd, seed )
    2l = area( seed ) / 2w
    // measure from the seed layer to the nwell layer
    // using the S pin. measure out to 200 um.
    ph = enclosure_perpendicular( seed, nwell, S, 200 )
    pv = enclosure_parallel( seed, nwell, S, 200 )

    // Calculations using built-in language exclusively
    pxh = sum( (ph::a + ph::b) * ph::w ) / 2w / 2
    pxv = sum( (pv::a + pv::b) * pv::w ) / 2l / 2
]
```

The same calculation can be carried out using a TVF\_NUMeric\_FUNction() (“[TVF Functional Interface for Device Property Calculation](#)” on page 1827). The TVF Functional Interface is generally faster than using the Device property enclosure functions with the sum() function, which was shown in the preceding example. An example with TVF\_NUMeric\_FUNction() is shown here:

```
device mp seed poly psd psd nwell [
    property 2w, 2l, pxh, pxv
    2w = perim_co( psd, seed )
    2l = area( seed ) / 2w
    // measure from the seed layer to the nwell layer
    // using the S pin. measure out to 200 um.
    ph = enclosure_perpendicular( seed, nwell, S, 200 )
    pv = enclosure_parallel( seed, nwell, S, 200 )

    // Same Calculations using TVF Function defined below
    pxh = TVF_numeric_function::device_func::calc_well_enc( ph, 2w )
    pxv = tvf_numeric_function::device_func::calc_well_enc( pv, 2l )
]

TVF FUNCTION device_func /*

proc calc_well_enc { enc w_or_l } {
    # functionally equivalent to sum( ph::a + ph::b ) * ph::w ) / 2w / 2
    # or sum( pv::a + pv::b ) * pv::w ) / 2l / 2
    set acum 0.0
    set slice_count [ $enc slice_count ]
    for { set i 0 } { $i < $slice_count } { incr i } {
        set acum [ expr { $acum + ( ( [ $enc a $i ] + [ $enc b $i ] ) *
            [ $enc w $i ] ) } ]
    }
    return [ expr { $acum / [ $w_or_l ] / 2 } ]
}

*/
```

## Example of Well Proximity Calculation

This shows how to measure proximity to an adjacent well, as opposed to an enclosing well. For example, proximity of an N-type device to adjacent nwell polygons.

Size the transistor gate by the *constraint-distance*, subtract the nwell from that, and use the result as the *measurement-pin-or-layer*.

```
// Typical CMOS layer derivation.  
bulk = extent           // Chip extent.  
psub = bulk not nwell  // P substrate.  
tran = poly and diff    // All transistor gates.  
sd = diff not poly     // Source-drain.  
ngate = tran and nplus // N gates.  
nsd = sd and nplus     // N source-drain.  
  
// Measurement layer derivation.  
ngate_s = size ngate by 7 // 7 is the constraint distance.  
mmmm = ngate_s not nwell // Measurement layer.  
  
device mn ngate poly nsd psub <mmmm> [  
    ...  
    ph = enclosure_perpendicular ( ngate, mmmm, S, 7 )  
    pv = enclosure_parallel ( ngate, mmmm, S, 7 )  
    ...  
]
```

The sizing operation is not strictly necessary; instead, you could just use psub as the measurement layer, since psub is chip extent minus nwell. However, with this procedure, you may reduce the amount of hierarchical degradation (known as seed promotion) caused by the operation.

## Example of Calculation for a Multifinger Enclosure

This example is similar to “[Example of Weighted Average Enclosure Calculation](#)” on page 1834. The property mf\_pxv is a weighted average of nwell enclosure over all gate fingers in the vertical direction. The final property is a mean value of all finger measurements. Remember that multifinger function output can only be processed by a Device TVF\_NUMeric\_FUNction() (see “[TVF Functional Interface for Device Property Calculation](#)” on page 1827).

```
DEVICE MN seed poly(g) a_pin(s) a_pin(d) <well> BY NET [  
    property 2w, 2l, mf_pxv  
    2w = perim_co( a_pin, poly )  
    2l = area( poly ) / 2w  
    pv = enclosure_parallel_multifinger( poly, well, s, 100 )  
    mf_pxv = TVF_numeric_function::device_func::mf_calc_well_enc( pv, 2l )  
]  
  
TVF FUNCTION device_func /*  
  
    proc mf_calc_well_enc { mf_enc 1 } {  
        set acum 0.0  
        set nfing [ $mf_enc finger_count ]  
        for { set j 0 } { $j < $nfing } { incr j } {
```

```

        set fing_acum 0.0
        set slice_count [ $mf_enc slice_count $j ]
        for { set i 0 } { $i < $slice_count } { incr i } {
            set fing_acum [ expr { $fing_acum + ( ( [ $mf_enc a $j $i ]
                + [ $mf_enc b $j $i ] ) * [ $mf_enc w $j $i ] ) } ]
        }
        set acum [ expr { $acum + $fing_acum } ]
    }
    return [ expr { ($acum / $nfing) / [$1] / 2 } ]
}

*/

```

## Performing Well Enclosure or Stress Effect Calculations with DFM Property

For performance reasons, it is frequently better to use the [device::enclosure\\_measurements](#) than the method described in this section because device::enclosure\_measurements is more efficient. [Table 5-8](#) shows differences between the relevant enclosure functions.

The [ENClosure\\_PARallel](#) and [ENClosure\\_PERpendicular](#) Functions previously described have two drawbacks. First, because they are part of the [Device](#) operation itself, they are limited to measuring features on the polygons that make up the device instance and not other nearby objects. Second, if large polygons are included, like well or diffusion shapes, calculations can become very time-consuming.

The [DFM Property](#) operation provides a mechanism in which DFM Property calculations are carried out using input from DRC operations. DRC operations take measurements of various device features. The DFM Property operation assigns the output of DRC operations to properties, which are then associated with specific device shapes. This allows the measurements to be carried out efficiently and separately from the Device computations, and allows the Device property computation program to access the measurements that pertain to a particular device.

---

### **Note**



The DFM Property statement used in these examples uses the Calibre nmDRC-H product license. Refer to the [Calibre Administrator's Guide](#) for exceptions.

---

Using layers derived from a DFM Property operation in a Device statement requires a Calibre Advanced Device Properties (ADP) product license.

---

The process of using DFM Property and Device statements to do enclosure property computations consists of three steps:

1. Write DRC error-directed dimensional check operations. The output of error-directed DRC operations is clusters of edges from input layers that meet the constraints of the DRC operations. Typically, these are Enclosure layer operations that measure distances needed for Device property computations.
2. Collect DRC measurements on polygon layers using the DFM Property operation.

3. Access DFM Property measurements in the Device property computation program.

## Method

Consider the following use of `enclosure_perpendicular` to collect gate-to-well dimensions perpendicular to the gate/source pin boundary:

```
SCH = ENCLOSURE_PERPENDICULAR(ngate, pwell, S, 5)
```

In order to write a DFM Property operation that mimics this calculation, each of the three steps outlined previously is carried out:

1. Write DRC error-directed dimensional check operations.

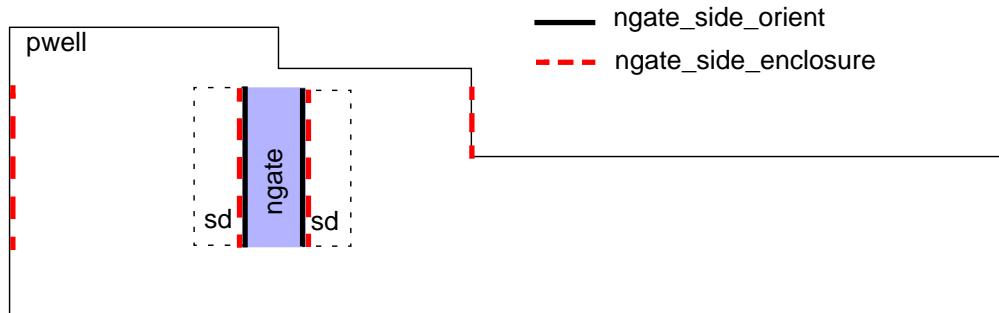
In order to make a similar calculation using the [DFM Property](#) interface, the orientation boundary between source/drain pins and the gate pin must be obtained. Once the edges of the gate/sd pin boundary are available, it is necessary to perform the measurements that collect the desired DFM Property values and associate them with a unique device. We are measuring the distance from the sd/ngate boundary to the pwell layer:

```
VARIABLE wproxrd 5 //well proximity background distance

// Perpendicular ngate orientation edges
ngate_side_orient = COINCIDENT EDGE ngate sd

ngate_side_enclosure = ENC ngate_side_orient pwell < wproxrd OPPOSITE
```

A device instance with these derived layers might appear like this:



2. Collect nmDRC measurements on polygon layers using DFM Property

Now you can use the DFM Property operation to gather measurements from the `ngate_side_enclosure` layer and to make properties out of them. These properties can then be associated with a unique device. These properties are assigned to the property name PER in the example shown in this section. The operation uses several DFM Property functions:

`ECMIN()`, `ECMAX()`, and `EW()` — to gather edge endpoints and spacings.

`VECTOR()` — to assemble the edge data into a vectors of edge start and end points, and a distance between edges.

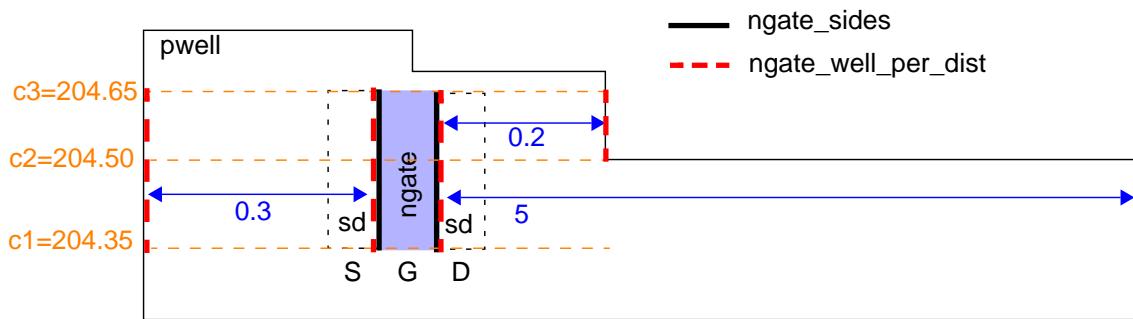
**CONCAT()** — to combine the distance-to-well measurements with a background measurement of 5 (wproxd variable) user units.

**SORT\_MERGE\_XXY()** — to arrange the two sets of edge measurements in the vectors into a single set of measurements that describe a measurement region around the device instance.

**RANGE\_XXY()** — to filter vector data to a specific interval. This function should always be used with SORT\_MERGE\_XXY().

These and other DFM Property functions are fully described in the *Calibre YieldAnalyzer and YieldEnhancer Reference Manual* sections entitled “Measurement Functions” and “Vector Expressions.”

Consider the following instance:



The DFM Property operation that calculates this configuration might look like this:

```

ngate_side = DFM PROPERTY ngate_side_orient
              ngate_side_enclosure
              OVERLAP ABUT ALSO MULTI SPLIT
[ PER = RANGE_XXY( SORT_MERGE_XXY( CONCAT(
              VECTOR( ECMIN(ngate_side_enclosure),
              ECMAX(ngate_side_enclosure),
              EW(ngate_side_enclosure)
            ), // measure any error clusters
              VECTOR( ECMIN(ngate_side_orient),
              ECMAX(ngate_side_orient),
              wproxd // 5 in VARIABLE statement
                  // defined previously.
            ) // set maximum background distance.
          ) // end CONCAT
        ), // end SORT_MERGE_XXY
        VECTOR( ECMIN(ngate_side_orient),
              ECMAX(ngate_side_orient))
      ) // end RANGE_XXY
]

```

The DFM Property vectors as delivered by **SORT\_MERGE\_XXY** are arranged as a series of width and offset pairs of the form  $\langle c_1, w_1 \rangle, \langle c_2, w_2 \rangle, \dots, \langle c_n, 0 \rangle$ . The  $c_1, c_2, \dots, c_n$  values represent the coordinates parallel to the error edge, and the  $w_1, w_2, \dots, 0$  values represent the distance between a particular pair of measurement

edges. Notice the final w value is always 0. For the preceding example, the vector returned for the source pin (S) is <204.35,0.3>,<204.65,0> and the vector returned for the drain pin (D) is <204.35,5>,<204.50,0.2>,<204.65,0>.

Finally, you need to attach the properties (which are currently attached to edges on the ngate\_side layer) to pin polygons that can be passed into the Device statement:

```
npin_prop = DFM PROPERTY ngate ngate_side
OVERLAP ABUT ALSO
[ PER1 = VPROPERTY( ngate_side, PER, 1 ) ]
[ PER2 = VPROPERTY( ngate_side, PER, 2 ) ]
```

The VPROPERTY function selects one of the property-assigned edges from the two ngate\_side edges that interact with the ngate polygon. The first selects edge “1”, and the second selects edge “2” using the third argument to VPROPERTY. These two properties are assigned the names PER1 and PER2. Note that they are not easily associated with either the S or D pin inside the Device built-in program specifically, but either one comes from either of the source and drain pins.

### 3. Access DFM Property measurements in the Device property program

The DFM Property vector created previously is accessed inside the Device property computation program using either the dfm\_vector\_value() function or the DFM\_VECtor\_ARRay() function, which are described under [“DFM Property Data Retrieval Functions”](#) on page 1788.

```
DEVICE MN ngate ngate(G) sd(S) sd(D) pwell(B) <npin_prop>
[
    property SEFFA_DFM, SEFFB_DFM

    // Note that DFM PROPERTY produces database unit measurements while
    // the Enclosure operation returns user units.
    //
    // Adjustment by UNIT_LENGTH() may be required to convert the
    // measurements to values expected by downstream tools.
    W = perimeter_coincide(ngate, sd)
    L = area(ngate)/W

    A_DFM = dfm_vector_value( npin_prop, "PER1" )
    B_DFM = dfm_vec_val( npin_prop, "PER2" )
    SEFFA_DFM = TVF_numeric_function::device_func::calc_seff( A_DFM, W,
    L, UNIT_LENGTH() )
    SEFFB_DFM = tvf_num_fun::device_func::calc_seff( B_DFM, W, L,
    unit_length() )

]
```

This Device statements makes a call to a TVF\_NUMeric\_FUNction() called device\_func, which contains a proc called calc\_seff. This is shown next:

```
TVF FUNCTION device_func /*
    package require CalibreLVS_DEVICE_DFM

    proc calc_seff { enc_prop W L unit_length } {
```

```
    set scale_factor [ $unit_length ]
    set enc_list [ device::scaled_dfm_vec_measurements $enc_prop
$scale_factor ]
    set accum 0.0
    set half_L [ expr { 0.5 * [$L] } ]
    foreach segment $enc_list {
        foreach { width length } $segment {
            set seg [ expr { ($width / ( $length + $half_L )) } ]
            set accum [ expr { $accum + $seg } ]
        }
    }
    return [ expr { ( [$W] / $accum ) - $half_L } ]
}

*/]
```

The Device TVF functional interface is used for these examples. This interface is described under “[TVF Functional Interface for Device Property Calculation](#)” on page 1827. Details about DFM Property vector objects and data-retrieval methods are given with the commands that produce the objects.

**Limitations** — There are some limitations to the use of DFM Property with Device property calculations:

- Device pushdown can be severely constrained when properties are attached to swappable pins.
- Measurements occur only between edges that are either horizontal or vertical and parallel to each other.

## Handling of Shorted Swappable Pins Using DFM Property

When a device has shorted, swappable pins, and the number of pin polygons touching the device seed shape is the same as the number of swappable pins, the DFM properties are assigned arbitrarily to one of each of the swappable pins. This is in contrast to the handling of other device properties such as AREA() and PERIMETER() function calculations where the cumulative calculation is assigned to one pin and the other pins are assigned a value of 0.

## Expression Evaluation Failures in DFM Property

When using DFM Property, it is important to protect against *expression evaluation failure*. Expression evaluation failure occurs in any situation where DFM Property cannot assign an appropriate property to polygons. Examples include numerical errors, lookup of non-existent properties, out-of-range properties, and so forth. Use of fall-through expressions in the DFM Property operation to account for expressions that could lead to expression evaluation failure is needed. See “[DFM Property in ADP Device Recognition](#)” in the *Calibre Verification User’s Manual* for a complete discussion.

## Device Annotation Built-In Language

The [Device Layer](#) operation supports DFM property device annotations through the ANNOTATE keyword and a program. A user-defined program is useful for adding measurements like stress effects that have the potential to disrupt the hierarchy of more exotic devices.

The annotation program uses a built-in language similar to the [Device Property Computation Built-In Language](#). The device annotation built-in language has all the features of the built-in languages described in [Table 5-1](#) and [Table 5-2](#). The specific aspects of the annotation language (sometimes referred to simply as the built-in language) are described in the following subsections.

Only one annotation program is allowed per Device Layer operation. Annotation specifications cannot be shared by multiple Device Layer operations.

These topics are covered in this section:

Writing a Device Annotation Program . . . . .	1842
Device Annotation Language Characteristics . . . . .	1843
Data Sources . . . . .	1844
DEBUG Statement . . . . .	1845
SELECT Statement . . . . .	1845
PROPerty Statement . . . . .	1846
Summary of Device Annotation Functions . . . . .	1847
Using DFM Property with Device Layer ANNOTATE Programs . . . . .	1848

## Writing a Device Annotation Program

Suppose you want to annotate a property value from a property named P1 to device seed shapes classified by the following DEViCe operation:

```
DEVICE mp(p) pgate1 pgate1(g) sd(s) sd(d) bulk1(b) <aux1> <aux2>
```

Using the built-in language, the [Device Layer](#) operation could look like this:

```

1  DEVICE LAYER mp(p) ANNOTATE dfm_layer
2  [
3      SELECT aux1, aux2
4      PROPERTY P1
5      P1 = DFM_NUM_VAL( dfm_layer, "P1" )
6  ]
```

This example illustrates many of the features of the language. The line numbers on the left are not part of the language. The language is case-insensitive.

Line 1 — Contains all of the Device Layer operation, except the annotation specification, which is contained between the matching square brackets on lines 2 and 6. The dfm\_layer parameter is a layer on which DFM properties are found and intersects the device seed shapes.

Line 3 — Contains the optional SELECT statement that declares layers that must be present in addition to dfm\_layer in order for a device seed shape to be selected by this Device Layer operation. The SELECT layers must be seed, pin, or auxiliary layers. In this case, they are auxiliary layers. The SELECT function acts as a filter on the device seed shapes returned by the Device Layer operation.

Line 4 — Contains a required PROPERTY function. This function defines which properties are annotated to the selected devices.

Line 5 — Assigns the value of property P1 from the dfm\_layer to the device shapes. The DFM\_NUM\_VAL() function selects the numeric value of DFM property P1 from dfm\_layer and assigns it to property P1, which is then annotated to all selected device shapes as a DFM property.

## Device Annotation Language Characteristics

The details of the device annotation built-in language are described in the following sections. Additional examples are provided. The following table shows information specific to the built-in language, which is in addition to the information provided in [Table 5-1](#) and [Table 5-2](#).

**Table 5-9. Device Annotation Built-In Language Specifics**

Element	Description
Structure	<p>The built-in language has a required sequence of statements, shown here:</p> <pre>[   DEBUG Statement    // optional, must be first if present.   SELECT Statement   // optional, must appear before PROPerTy.   PROPerTy Statement // required.   annotation computations // the properties declared in the                           // PROPerTy statement must be                           // calculated in the program. ]</pre> <p>There is no WARN statement in this language. Refer to “<a href="#">Structure</a>” in <a href="#">Table 5-1</a> for general notes on the structure of the language.</p>
Macros	Annotations can be written as DMACRO statements that can be called multiple times in a rule file.
Local Variables	You can assign intermediate values to local variables within an annotation specification. Numeric, string, and complex type local variables are supported. Types for these variables are determined by the functions that are used. Once a type is associated with a variable, the type may not change.

**Table 5-9. Device Annotation Built-In Language Specifics (cont.)**

Element	Description
Assignment Statements	<p>Assignment statements have the following forms:</p> $\text{local\_variable\_or\_property\_name} = \text{numeric\_expression}$ $\text{local\_variable\_or\_property\_name} = \text{string\_expression}$ $\text{local\_variable\_or\_property\_name} = \text{function}$ <p>The last case is for functions that return complex-type variables. You can specify only one = operator per assignment statement.</p> <p>Note: Variables types cannot be changed after the initial assignment.</p> <p>Examples:</p> <pre>A = 5                                     //numeric C = "ABC"                                  //string E = DFM_VECTOR_VALUE( sd_prop, "NC" ) //array</pre>
Units of measurement	<p>For information about which units to use for representing physical quantities such as length and area, refer to “Units of Measurement” in the <i>Calibre Verification User’s Manual</i>.</p>
TVF interface	<p>The <a href="#">TVF Functional Interface for Device Property Calculation</a> is governed by the rules of Tcl code, which are somewhat different from the Device property computation built-in language. When working with TVF Function blocks, ensure that you use valid Tcl code.</p>
Pre-processor directives	<p>Pre-processor directives, or conditionals, are permitted in device annotation calculations. These can help you to control which properties get calculated for the type of run you are using. See “<a href="#">Pre-Processor Directives</a>” on page 56.</p>

## Data Sources

Data used in the computation can come from any of these sources:

- Constants — You can use numeric, floating-point constants, and string literals directly within the property specification.
- Numeric Process Variables — These values are accessed by using a rule file [Variable](#) just as any other variable within the property specification.
- String Process Variables — Rule file variables may be assigned to property variables (names declared in the PROProperty statement) or assigned to local variables. No other use of string rule file variables is supported.

Rule file variables may never appear as the left-hand side of an assignment. Doing so results in a compiler error.

- Instance Data — Geometric, connectivity, and text data associated with an instance are accessed by the provided built-in functions. Properties generated on specific shapes using a **DFM Property** operation are available to commands that support them. A complete list of the functions and their definitions is given under “[Summary of Device Annotation Functions](#)” on page 1847.

## DEBUG Statement

Tracing of the annotation computation can be useful in finding errors during the development of rule file code. Tracing is controlled by the DEBUG statement, which is optional. If present, it must be the first statement in the program. It has the following form:

**DEBUG *range1* [ , *range2* , ... *rangeK* ]**

where each range is either a device instance number, or a pair of device instance numbers separated by a hyphen (-). At least one range must be specified, although it can contain only a single instance. For example, the statement:

`DEBUG 0-2, 50`

causes a step-by-step report of the annotation computation to be printed for each of the device instances 0, 1, 2, and 50. The trace can produce much output, so the use of small ranges is recommended. You can obtain the instance numbers from LVS discrepancy reports in nmLVS-H.

## SELECT Statement

The optional SELECT statement defines seed, pin, or auxiliary layers that must be present on a device in order for an annotation to apply. In other words, this statement filters devices based upon the presence of the specified layers. When used, this statement must follow a DEBUG statement (if used) and precede a PROPerity statement. The statement has this form:

**SELECT *layer* [, *layer*] ...**

At least one layer must be specified. Multiple SELECT statements are allowed and they are each applied individually.

Consider the following set of device statements:

```
DEVICE MP(p) PGATE1 PGATE1(g) SD(s) SD(d) BULK1(b) <AUX1> <AUX2> // $D0
DEVICE MP(p) PGATE2 PGATE2(g) SD(s) SD(d) BULK1(b) <AUX1> // $D1
DEVICE MP(p) PGATE1 PGATE1(g) SD(s) SD(d) BULK1(b) // $D2
DEVICE MP(p) PGATE2 PGATE2(g) SD(s) SD(d) BULK1(b) // $D3
```

The Device Layer operation:

```
X = DEVICE LAYER mp(p) ANNOTATE dfm_layer [
    PROPERTY p1
    p1 = DFM_NUM_VAL( dfm_layer, "P1" )
]
```

returns shapes from all \$D templates that interact with dfm\_layer and transfers the DFM property P1 from dfm\_layer onto the resulting shapes.

The SELECT statement can be used to choose specific device templates that contribute to the result layer:

```
X = DEVICE LAYER mp(p) ANNOTATE dfm_layer [
    SELECT aux1, aux2
    PROPERTY p1
    p1 = DFM_NUM_VAL( dfm_layer, "P1" )
]
```

The statement SELECT aux1, aux2 adds any device that has both layers aux1 and aux2 as a required seed, pin, or auxiliary layer. In this example, it will choose just the first (\$D0) device because only those devices have both aux1 and aux2 layers.

Multiple SELECT statements each contribute additional devices to the result layer:

```
X = DEVICE LAYER mp(p) ANNOTATE dfm_layer [
    SELECT aux1, aux2
    SELECT pgate2
    PROPERTY p1
    p1 = DFM_NUM_VAL( DFM_LAYER, "P1" )
]
```

The statement SELECT pgate2 adds to the original selection of \$D0. It contributes seed shapes from both \$D2 and \$D4, because both of these templates have pgate2 in them.

## PROPerty Statement

The PROPerty statement declares the names of the properties to be computed. This statement is required and, in the absence of the DEBUG or SELECT statements, must be the first statement in a property computation. It has the following form:

**PROPerty prop\_name\_1, prop\_name\_2, ...**

In the following example, four properties are declared with names W, L, SA, and DA:

```
PROPERTY W, L, SA, DA
```

Property names are treated as variables within computational statements. The final value assigned to the property name is the value of the DFM property associated with the instance.

Property names must begin with a letter character followed by any legal SPICE characters. See “[General SPICE Syntax Summary](#)” in the *Calibre Verification User’s Manual* for more details. The rule file compiler will allow you to specify property names that are not SPICE compliant, so care should be taken in choosing names. The Calibre nmLVS SPICE reader will complain about illegal property names that appear in a netlist generated from bad property names in the rule file.

## Summary of Device Annotation Functions

A variety of built-in functions provide access to geometric and connectivity information about an instance. You can use these functions within numeric, string, or complex expressions. Numeric results can be used as arguments to relational operators.

Many of the functions take one or more arguments that are references to either a layer or a pin. When referencing a pin, the compiler uses the name specified in the Device operation. When referencing a layer, the compiler uses the name or number of the layer only if it matches the name or number in the layer list of the Device operation. If the reference could be interpreted as either a pin or a layer, the pin interpretation is chosen.

Table 5-10 lists the data retrieval functions available for the built-in language, along with a description of their use.

**Table 5-10. Device Annotation Built-In Functions**

Function	Description
<b>Device Property Function</b>	In place of the measurement functions of the Device property computation built-in language, this function accesses Device property values directly.
<a href="#">device_numeric_value</a>	Provides access to properties calculated in Device statements.
<b>DFM Property Data Retrieval Functions</b>	These functions are used in conjunction with the DFM Property operation to retrieve property values for use within Device property computations. Refer to “ <a href="#">DFM Property Data Retrieval Functions</a> ” on page 1788.
<a href="#">dfm_numeric_value</a>	Provides access to a DFM property from a single polygon on a pin, pin layer, or auxiliary layer. The DFM property is accessible as a regular numeric value.
<a href="#">dfm_vector_value</a>	Provides access to a DFM vector property from a single polygon on a pin, pin layer, or auxiliary layer. The set of properties is accessible through a TVF object created by this function.
<b>Net function</b>	This function is useful for detecting pins that are connected to special nets such as power or ground.
<a href="#">pin_net</a>	Returns the net number of a specified pin. In hierarchical applications, this function returns node number local to the cell, which is a limitation.
<b>Unit functions</b>	These return units of measure.
<a href="#">precision</a>	Returns the current value of the process precision (1000 by default).
<a href="#">unit_capacitance</a>	Returns the current value of the unit capacitance as specified in the rule file.

**Table 5-10. Device Annotation Built-In Functions (cont.)**

<b>Function</b>	<b>Description</b>
<a href="#">unit_length</a>	Returns the current value of the process <a href="#">Unit Length</a> , 1E-6 by default.
<a href="#">unit_resistance</a>	Returns the current value of the unit resistance as specified in the rule file.
<b>Device TVF functions</b>	The Tcl Verification Format interface for Device property calculations is described under “ <a href="#">TVF Functional Interface for Device Property Calculation</a> ” on page 1827.
<a href="#">TVF_numeric_function</a>	Provides access to a numeric value returned by a Tcl procedure defined within a TVF Function definition.
<a href="#">TVF_string_function</a>	Provides access to a string value returned by a Tcl procedure defined within a TVF Function definition.

The following command is the only one unique to the Device Annotation Built-In Language. The remainder of the functions are defined in the “[Function Reference](#)” on page 1786.

## device\_numeric\_value

**DE**vice\_NUMeric\_VALue(*property\_name*)

### Description

Returns the numeric value associated with the *property\_name*. The *property\_name* is expected to be numeric and calculated by the Device operation.

### Example

Consider the following Device statement:

```
DEVICE mp(p) pgate pgate(g) sd(s) sd(d) bulk(b)
```

By default, properties L and W are calculated for built-in MOS devices. You could use this code to access the L property in a device annotation program:

```
X = DEVICE LAYER mp(p) ANNOTATE dfm_layer [
    PROPERTY p
    p1 = DFM_NUM_VAL( dfm_layer, "P1" ) // DFM property value
    L = DEV_NUM_VAL( L ) // DEDevice property value
    p = p1 * L
]
```

Any property that is calculated for a device can be accessed by DEDevice\_NUMeric\_VALue using a similar method.

## Using DFM Property with Device Layer ANNOTATE Programs

DFM Property provides a powerful mechanism for collecting and combining complex measurements. The [Device Annotation Built-In Language](#) provides a mechanism for combining

DFM Property measurements with devices recognized using the Calibre nmLVS device recognition facility.

Netlists that combine these calculated properties with the originally recognized devices are available through the Query Server CCI interface. See “[Annotated Device Commands](#)” in the *Calibre Query Server Manual* for details.

The following example recognizes devices, attaches DFM Property measurements to the devices using the Device Layer ANNOTATE option, and creates a netlist of the new properties using the Query Server’s CCI interface. This illustration calculates AS and AD, area of the source and drain pins, for an N-type MOSFET:

```
// First gather the area property from the source/drain pin layers:
NSD_wArea = DFM PROPERTY NSD [ A = AREA( NSD ) ]

// Next, accumulate pin area and pin net information onto the device seed
shape

NGate_all_asad = DFM PROPERTY (DEVICE LAYER MN) NSD_wArea OVERLAP ABUT
ALSO MULTI
[ N1 = NETID(NSD_wArea, 1) ]

// Total area for the first net (full area if pins are shorted)
[ C = COUNT(NSD_wArea) ]
[ AN1 = VNETSUM( NETID( NSD_wArea, 1 ),
                  VNETID( NETID( NSD_wArea ) ),
                  VECTOR(PROPERTY( NSD_wArea, A ))) ]
]

// Area for the first pin polygon
[ A1 = PROPERTY( NSD_wArea, A, 1 ) ]

// Total area for all pin nets
[ ATotal = PROPERTY( NSD_wArea, A ) ]

// Finally, transfer the AS and AD properties onto the recognized device.

DLAYER_MN_PROP = DEVICE LAYER MN(n) ANNOTATE NGate_all_asad
[
    PROPERTY AS, AD

    if ( PIN_NET(S) == PIN_NET(D) ) {
        // Shorted Source/Drain
        // This mimics the distribution done by Device Recognition
        if ( DFM_NUM_VAL(NGate_all_asad, "C") == 2 ) {
            // This is arbitrary
            AS = DFM_NUM_VAL(NGate_all_asad, "A1")
            AD = DFM_NUM_VAL(NGate_all_asad, "ATotal") - AS
        } else {
            AS = DFM_NUM_VAL(NGate_all_asad, "ATotal")
            AD = 0
        }
    } else {
        if ( PIN_NET(S) == DFM_NUM_VAL(NGate_all_asad, "N1") ) {
            AS = DFM_NUM_VAL(NGate_all_asad, "AN1")
            AD = DFM_NUM_VAL(NGate_all_asad, "ATotal") - AS
        }
    }
]
```

```
        } else {
            AD = DFM_NUM_VAL(NGate_all_asad, "AN1")
            AS = DFM_NUM_VAL(NGate_all_asad, "ATotal") - AD
        }
    ]
}
```

Add these statement- to the rule file:

```
LVS ANNOTATE DEVICES Ngate_all_asad
MASK SVDB DIRECTORY svdb QUERY CCI
```

When you run netlist extraction, the devices are written without annotation properties. To get a netlist with the annotation properties, you run Query Server CCI on the output.

After opening the SVDB directory in Query Server, you can then run CCI commands such as this:

```
GDS ANNOTATED DEVICES YES
GDS WRITE annotated_MN.agds
LAYOUT NETLIST ANNOTATED DEVICES YES
LAYOUT NETLIST WRITE annotated_MN.sp
TERMINATE
```

These commands generate an annotated GDS file and layout netlist, respectively.

## Device Reduction Effective Property Computation Language

You can specify an device reduction effective property computation program in most LVS device reduction statements (all begin with “LVS Reduce”). This optional section consists of a user-defined program that defines the calculations to take place during device reduction. It has all the features of the built-in languages described in [Table 5-1](#) and [Table 5-2](#).

You can specify only one effective property program per LVS device reduction specification statement. The program lists the names and types of the properties to be computed upon reduction and specifies the method of computation from available data. The effective property program is placed between a pair of square brackets at the end of an LVS device reduction specification statement. This placement is an extension of the use of square brackets to contain reduction TOLERANCE statements. Effective property programs cannot be shared by multiple device reduction specification statements.

The presence of an effective property program cancels and overrides any built-in effective property computation for the particular device type. When you provide an effective property program, you must define formulas for all properties of interest, including built-in properties. Some effective property programs can become lengthy and may extend over several lines in the rule file.

You can specify an effective property program in the following specification statements:

<a href="#">LVS Reduce Parallel Bipolar</a>	<a href="#">LVS Reduce Series Capacitors</a>
<a href="#">LVS Reduce Parallel Capacitors</a>	<a href="#">LVS Reduce Series MOS</a>
<a href="#">LVS Reduce Parallel Diodes</a>	<a href="#">LVS Reduce Series Resistors</a>
<a href="#">LVS Reduce Parallel MOS</a>	<a href="#">LVS Reduce Split Gates</a>
<a href="#">LVS Reduce Parallel Resistors</a>	<a href="#">LVS Reduce</a>

You *cannot* specify an effective property program in the following statement:

[LVS Reduce Semi Series MOS](#)

The default effective property computations are shown in the “Device Reduction” section of the [Calibre Verification User’s Manual](#). Device properties are discussed with the [Device](#) command.

## Writing an Effective Property Computation Program

This section describes a sample usage of the effective property language by explaining the example line-by-line.

This example illustrates many of the features of the language. The line numbers on the left are not part of the example specification and the language is case-insensitive.

In the explanation that follows the example, the term *input group* refers to the set of devices being reduced to a single instance. Effective property values are calculated based upon this input group, and stored on the resulting (*effective*) instance.

```

1  lvs reduce parallel resistors yes
2      [
3          effective r, l, w
4          //
5          p = sum (w*l)
6          q = sum (w/l)
7          w = sqrt (p*q)
8          l = sqrt (p/q)
9          //
10         checkForZero = prod (r)
11         if (checkForZero == 0)
12             r = 0
13         else
14             { // demonstrate braces
15                 // assign 1/sum(1/r) in two steps:
16                 recipSum = sum(1/r)
17                 r = 1 / recipSum
18             }
19     ]

```

Line 1 shows a standard LVS Reduce specification statement. In this case, it enables parallel resistor reduction.

Line 2 has an opening square bracket. The opening square bracket ( [ ) is necessary to indicate an effective property program. This square bracket must immediately follow the YES keyword (there may be only white space between YES and the bracket). The square bracket may only be present if YES is present.

Line 3 shows the EFFECTIVE property statement. The EFFECTIVE property statement declares the names of all properties for which effective property values are computed throughout the program. Each property listed here must be assigned a value in the program. If LVS is to calculate an effective property, you must list the property in this statement.

In this example, the names represent both input values (properties found on the input group) and output values (the properties to be calculated and assigned to the resulting instance).

The properties are part of a comma (,) separated list.

Line 4 is a commented line.

Lines 5 and 6 show local variable declaration and value assignment.

Line 5 declares the local variable P and assigns the value:

$$W_1 \cdot L_1 + W_2 \cdot L_2 + \dots + W_n \cdot L_n$$

where  $W_i$  and  $L_i$  are the width and length of the  $i$ th device in the input group, respectively.

Line 6 declares the local variable Q and assigns the value:

$$W_1 / L_1 + W_2 / L_2 + \dots + W_n / L_n$$

where  $W_i$  and  $L_i$  are the width and length of the  $i$ th device in the input group, respectively.

The W and L variables in these equations are not the same variables in lines 7 and 8. The W and L variables referred to here are retrieved from the input group, while the W and L variables in lines 7 and 8 are assigned to the resulting instance.

Lines 7 and 8 show value assignment to variables declared in the EFFECTIVE property statement.

Line 7 shows that the value W, declared in the EFFECTIVE property statement, is assigned a value.

Line 8 shows that the value L, declared in the EFFECTIVE property statement, is assigned a value.

Because W and L are declared in the EFFECTIVE property statement, they must be assigned values somewhere in the program. These two lines satisfy that requirement. The resulting W and L values are assigned to the single instance which represents the reduced input group.

Line 9 is a commented line.

Lines 10 through 18 demonstrate property language usage, showing how to avoid the use of zero (0) values from the input group.

If you provided only the statement:

```
r = 1/sum(1/r)
```

and an  $R_i$  in the input group had a zero value, the effective R value would be set to the unknown value. The unknown value is represented by a question mark (?) in the LVS report. The unknown value is assigned because division by zero is attempted and is undefined. The unknown value can cause property discrepancies to be reported when the [Trace Property](#) statement is in use. The unknown value can also cause input group reduction to be avoided when you specify the TOLERANCE keyword in an LVS device reduction statement. Instead, lines 10 through 18 set the effective R value to zero if the input group contains any zero-valued  $R_i$  values.

Line 10 declares the local and temporary variable checkForZero and assigns the value:

```
R1 * R2 * ... * Rn
```

where  $R_i$  represents the R value for the  $i$ th instance in the input group. Note that checkForZero is zero if *any* of the input R values are zero.

Line 11 begins a conditional statement.

Line 12 assigns the effective R value a zero value if the conditional statement in line 11 is true. This avoids the use of the unknown value.

Line 13 shows the ELSE statement.

Line 14 demonstrates the use of the opening brace ( { ) and how it can be used to group statements into a single logical statement. As in C or C++, this grouping is useful for indicating the extent of a dependent clause for a conditional statement.

Line 15 is a commented line.

Line 16 declares the local, temporary variable recipSum and assigns the value:

`1/R1 + 1/R2 + ... 1/Rn`

where  $R_i$  represents the R value for the  $i$ th instance in the input group.

Line 17 assigns the effective R value the value:

`1 / recipSum`

which is equivalent to

`1 / ( 1/R1 + 1/R2 + ... 1/Rn )`

You could replace lines 14 through 18 with the single statement:

`R = 1/SUM(1/R)`

but the purpose of this example is to show the use of statement grouping within braces ( { } ).

Line 18 contains a closing brace ( } ) that ends the statement group.

Line 19 contains a closing bracket ( ] ) that ends the effective property program.

The details of the effective property computation language are described in the following sections. More examples are provided.

## Effective Property Language Characteristics

The following table shows information specific to the effective property computation language, which is in addition to the information that appears in [Table 5-1](#).

**Table 5-11. Effective Property Language Specifics**

Element	Description
Structure	<p>The built-in language has a required sequence of statements, shown here:</p> <pre>[ // see <a href="#">EFFECTIVE and EFFECTIVE STRING Statements</a> // at least one of these statements must appear first in // the program. Both may be present, and they can be in // any order. Effective Property statement Effective String property statement  <a href="#">DEBUG Statement</a> // optional, must be after the Effective // and Effective String statements, and // before the Warn statement. <a href="#">WARN Statement</a> // optional, must be after the optional // DEBUG statement. property computations // the properties declared in the // Effective and Effective String // statements must be defined // in the program. ]</pre> <p>Refer to <a href="#">Structure</a> in <a href="#">Table 5-1</a> for general notes on the structure of the language.</p>
Input Group	Refers to the set of devices that are being reduced to a single instance. Effective property values are calculated based upon this input group and stored on the resulting ( <i>effective</i> ) instance.
Data Types	The effective property computation language handles both numeric and character string properties. All numeric literal values are specified in double precision floating point. Effective calculations specified by the user program are also carried out in double precision. Note, however, that effective values are restricted to single precision during the LVS comparison phase. Effectively, values are restricted to single precision when they are assigned to the single instance representing a reduced input group.
String Expressions	A string expression may consist only of a single string function reference. String-valued local variables are not supported. See “ <a href="#">Limitations on String Properties</a> .”

## Data Sources

Data used in the computation may come from any of the following sources:

- Numeric constants — You can use numeric floating-point constants directly within the program.
- Instance data — You can obtain property data associated with instances in the input group with built-in vector functions. Vector functions include the SUM() and PROD() functions. For a complete list of vector functions, refer to section “[Effective Property Language Vector Functions](#)” on page 1859.

When you specify a property name X inside a vector function, it refers to individual  $X_i$  values on the instances in the input group, treated as a vector or array. In all other cases, X is a scalar and refers to the current property value on the reduced or effective instance.

- Local variables — You can declare local variables, also referred to as temporary variables, and assign them scalar values. Local variables cannot serve as arguments to vector functions because there are no such property values associated with the instances in the input group. That is, there are no individual values for local variables associated per-instance in the input group.

## EFFECTIVE and EFFECTIVE STRING Statements

The EFFECTIVE or the EFFECTIVE STRING statements are used to declare the names of the properties for which effective values should be computed. The EFFECTIVE statement declares numeric properties and the EFFECTIVE STRING statement declares string properties (that is, properties with character string values). Either one or the other is required, and both may appear. The EFFECTIVE statement may precede or follow the EFFECTIVE STRING statement, but they must both precede the optional DEBUG and WARN statements. These statements have the following form:

**EFFECTIVE *property\_name* [..., *property\_name*]**

**EFFECTIVE STRING *property\_name* [..., *property\_name*]**

Properties listed here are considered both input (should be present on instances in the input group), and output (calculated values are assigned to the reduced instance).

Property names must be simple names; property-name/spice-parameter combinations such as “instpar(w)” are not accepted in effective property computation programs. To handle such combinations use the [LVS Property Map](#) specification statement.

In the following example, four numeric properties are declared with names W, L, AS and AD. Also, two string properties are declared with names xStr and yStr.

```
EFFECTIVE W, L, AS, AD
EFFECTIVE STRING xStr, yStr
```

A property name cannot be declared in both an EFFECTIVE and EFFECTIVE STRING statement within the same property computation program. For example, the following generates a compiler error:

```
LVS REDUCE ... [
    EFFECTIVE x
    EFFECTIVE STRING x
    x = MIN(x)
]
```

Property names are treated as variables within computational statements.

As mentioned in “[Data Sources](#)” on page 1856, a property name “X” used inside a vector function (such as SUM() or MIN()), refers to  $X_i$  values on the instances in the input group, treated as a vector. In all other cases, “X” is a scalar and refers to the current property value on the reduced or “effective” instance.

If you declare a property name in the EFFECTIVE statement, you must assign it a numeric value somewhere in the program. If you declare a property name in the EFFECTIVE STRING statement, you must assign it a string value somewhere in the program. If you do not, a compile error results.

When the program concludes, the value of each property is assigned to the single instance reduced from the input group.

Note that the presence of effective property computation in a device reduction program does not, on its own, trigger missing property errors when properties involved in the computation are missing from a particular input device. Only “actionable” property statements trigger missing property errors. Actionable property statements are, for example, TOLERANCE (within device reduction programs), [Trace Property](#), [LVS Filter](#) with string constraints, and [LVS Split Gate Ratio](#). In other words, if a property is not processed, directly or indirectly, by any “actionable” statement, then it does not trigger missing property errors.

## Limitations on String Properties

This section summarizes the various limitations involving string properties in the effective property computation language.

- String properties are supported in limited contexts. Specifically, string properties may only appear as the single argument to a string-valued function, and as the left-hand side of an assignment statement whose right-hand side is a string-valued function. No other operators are supported.

Properties declared as EFFECTIVE STRING may not appear in any numeric context.

The following example generates compiler errors:

```
EFFECTIVE STRING str
str = 1 + 1      // Error, trying to assign numeric
                  // expression to string.
x = str + 1     // Error, string variable appears in
                  // numeric context.
...
```

- The effective property computation language does not support string-valued local variables. Assignment of literal string values is not supported. Assignment of string valued variables to another variable is not supported. Use of any of these forms results in a compiler error. For example:

```
EFFECTIVE STRING str
str = "my_string"      // Error, no string literals.
s1  = str      // Error, no variable to variable string assignment.
```

## DEBUG Statement

This statement controls the debug tracing of the effective property computation. It is useful in finding errors during the development of new effective property computation code.

This is an optional statement. If you use it, you must specify it after the EFFECTIVE and EFFECTIVE STRING statements in the effective property program, and before a WARN statement. The DEBUG statement has the following syntax:

**DEBUG range1 [ , range2 , ... rangeK ]**

where each range is either an instance ID, or a pair of instance IDs separated by a hyphen (-). You must specify at least one range value, although it can contain a single instance ID.

An instance ID represents a single device to which the input group is reduced. Instance IDs are unique across all devices in the design (layout or source) being transformed. LVS assigns unique instance IDs. They are not necessarily sequential. Instance IDs are most useful in debugging warnings produced when the [WARN Statement](#) is present and an impossible mathematical operation is attempted.

The following is an example of a DEBUG statement:

```
DEBUG 0-2, 508
```

This statement prints a detailed report to the transcript of the property computation for each device instance with an ID of 0, 1, 2, and 508. The trace can produce a large amount of output, so the use of small ranges is recommended. You can obtain instance IDs from output produced by the WARN statement.

## WARN Statement

This optional statement prints a warning to the transcript if the effective property program attempts to perform an erroneous mathematical computation, such as dividing by zero or taking the square root of a negative number. LVS does not generate an error when performing an impossible mathematical computation.

If specified, it must appear after the optional DEBUG statement. The WARN statement has the following syntax and it takes no arguments:

**WARN**

If the tool performs any erroneous mathematical computations, the effective property is set to the unknown value, which results in it being represented by a question mark (?) in the LVS report. When a property is set to the unknown value, it can cause discrepancies when tracing properties. An unknown value also precludes devices in the input group from being reduced together if you specify TOLERANCE in the device reduction specification statement.

The following is a sample transcript of the warning printed by the WARN statement:

```
*****
EFFECTIVE property user-program WARNING:
Division by zero (result set to UNKNOWN).
while transforming layout with:
LVS REDUCE ... EFFECTIVE clause: (rule file line 58)
Use instance ID# 92 in DEBUG trace range.
*****
```

You can use the instance ID# 92 in a DEBUG trace range to see exactly how the unknown value was determined for this input group and the effective (reduced) instance.

## Effective Property Language Vector Functions

You use vector functions to evaluate an expression across all N instances in the input group. The resulting set of N expressions can be summed, multiplied, searched for a minimum, or searched for a maximum. You can count the number of instances in the input group with the COUNT() function.

This is the complete set of vector functions and their syntax:

```
count()
equal ( vector_expression )
max ( vector_expression )
min ( vector_expression )
prod ( vector_expression )
sum ( vector_expression )
```

where *vector\_expression* is required by all vector functions, except COUNT(). The expression can be numeric or string type.

### Numeric *vector\_expression* argument

A numeric *vector\_expression* is the same as any other numeric expression, except that it can only reference numeric property names and simple numeric constants. A property name is numeric if it is declared in a EFFECTIVE statement.

Vector functions are numeric when their *vector\_expression* argument is of numeric type. Numeric functions expect numeric property values in the input group and return numeric values. If a property value in the input group has the wrong type, the value “missing” is used in place of the original value.

For example, if you declared the property name W in the EFFECTIVE statement, then the following assignment:

```
W = PROD ( W+1 )
```

shows a valid *vector\_expression*. While the following assignment:

```
W = PROD ( W+sqrt(2) )
```

is not valid because W+sqrt(2) is not a valid numeric *vector\_expression*. It does not consist of only property names and simple numeric constants, but instead references another scalar function (sqrt()).

## String vector\_expression argument

A string *vector\_expression* may only consist of a single string-type property name. A property name is of type string if it is declared in a EFFECTIVE STRING statement.

If Z is a property name declared in the EFFECTIVE STRING statement then the only *vector\_expression* involving Z is the property name itself, “Z”. For example:

```
Z = MIN( Z )
```

Vector functions are string-type when their *vector\_expression* argument is of string type. String functions expect string property values in the input group and return string values. If a property value in the input group has the wrong type, the value “missing” is used in place of the original value.

Also see “[Example 6](#)” on page 1864.

## count

### COUNT()

Returns the number of instances present in the input group. This can be useful in calculating average values, as in the following example:

```
W = SUM(W) / COUNT()
```

## equal

### EQUAL (*vector\_expression*)

Evaluates *vector\_expression* across all N instances in the input group. If all such evaluations result in the same value, EQUAL() returns the numeric constant 1. If at least one of the evaluations of *vector\_expression* differs from the others, EQUAL() returns the numeric constant 0.

When a string *vector\_expression* is given to EQUAL(), that expression is compared for equality across all N instances in the input group. Case sensitivity during equality comparison is controlled by the setting of [LVS Compare Case](#).

Here is an example of the use of EQUAL():

```
LVS REDUCE type(subtype) PARALLEL [
    PROPERTY xprop
    PROPERTY STRING ypropStr
    all_xprop_plus_one_are_the_same = EQUAL(xprop+1)
    all_ypropStr_are_equal = EQUAL(ypropStr)
    ...
]
```

## max

### **MAX (vector\_expression)**

Evaluates *vector\_expression* across all N instances in the input group and returns the maximum expression value found. Either numeric or string arguments are allowed.

Recall that a string-type *vector\_expression* argument may only consist of a single string property name. For string-type *vector\_expression* arguments, the function selects the lexicographically largest string value found for the specified property in the input group (the group of devices being reduced together). Case sensitivity for comparisons is determined by the VALUES setting in the [LVS Compare Case](#) specification statement.

The following example:

```
L = MAX ( L )
```

returns the largest  $L_i$  found in the input group ( $L_i$  represents the value of L for the  $i^{\text{th}}$  instance in the group). Largest is defined numerically if L is numeric, or lexicographically if L is string-type.

## min

### **MIN (vector\_expression)**

Evaluates *vector\_expression* across all N instances in the input group, and returns the minimum expression value found. Either numeric or string arguments are allowed.

Recall that a string-type *vector\_expression* argument may only consist of a single string property name. For string-type *vector\_expression* arguments, the function selects the lexicographically smallest string value found for the specified property in the input group (the group of devices being reduced together). Case sensitivity for comparisons is determined by the VALUES setting in the [LVS Compare Case](#) specification statement.

The following example:

```
L = MIN ( L )
```

returns the smallest  $L_i$  found in the input group ( $L_i$  represents the value of L for the  $i^{\text{th}}$  instance in the group). Smallest is defined numerically if L is numeric, or lexicographically if L is string-type.

## prod

### **PROD (vector\_expression)**

Returns the product of *vector\_expression* applied to all N instances in the input group. Only numeric expressions are allowed.

The following example:

```
QQ = PROD ( W/L+1 )
```

returns:

$$(W_1/L_1+1) * (W_2/L_2+1) * \dots * (W_n/L_n+1)$$

## sum

### **SUM (vector\_expression)**

Returns the sum of *vector\_expression* applied to all N instances in the input group. Only numeric expressions are allowed.

The following example:

```
P = SUM ( W*L )
```

returns:

$$W_1*L_1 + W_2*L_2 + \dots + W_n*L_n$$

## Default Reduction of String Properties

When no effective property computation section is present in device reduction specification statement, string properties are reduced as follows:

- If all values for a particular string property are equal across the input group of instances, then the reduced instance is given the single string value observed.
- Otherwise, the value “unknown” is assigned.

Case sensitivity for comparisons is determined by the VALUES setting of the [LVS Compare Case](#) specification statement.

## Effective Property Computation Examples

The following rule file examples illustrate effective property computations:

### Example 1

```
LVS REDUCE PARALLEL MOS YES [
    effective W           // Reduce MOS devices in parallel.
    W = sum( W )          // Calculate effective width value
]                         // as sum of original widths,
                           // regardless of length.
```

## Example 2

```
LVS REDUCE PARALLEL MOS YES [
    // Reduce MOS devices in parallel only if
    tolerance L 0           // equal lengths.
    effective W, L          // Calculate effective width and length.
    W = sum( W )             // Width is sum of original widths.
    L = min( L )             // OK to use minimum since all are equal.
]
```

## Example 3

```
// This is equivalent to the default built-in effective
// property calculation for parallel MOS devices.
//
LVS REDUCE PARALLEL MOS YES [
    // Reduce MOS devices in parallel.
    effective W, L, AS, AD, PS, PD
    // Calculate these effective values.
    P = sum( W * L )          // Sum of Wi * Li
    Q = sum( W / L )          // Sum of Wi / Li
    W = sqrt( P * Q )         // Effective W
    L = sqrt( P / Q )         // Effective L
    AS = sum( AS )            // Effective AS: sum of ASi
    AD = sum( AD )            // Effective AD: sum of ADi
    PS = sum( PS )            // Effective PS: sum of PSI
    PD = sum( PD )            // Effective PD: sum of PDI
]
```

## Example 4

```
// Reduce resistors in parallel only when all W values are
// equal. Effective W is equal to the original W (in a sense,
// W describes the type of resistor).
//
LVS REDUCE PARALLEL RESISTORS YES [
    tolerance W 0
    effective W
    W = min( W )             // OK to use minimum since all are equal
]
```

## Example 5

```
LVS REDUCE PARALLEL RESISTORS YES [
    // Reduce parallel resistors only if equal width
    tolerance W 0
    effective R, W
    // Calculate effective resistance and width
    R = 1 / sum( 1/R ) // Same as the default calculation
    W = min( W )       // OK to use minimum since all are equal
]
```

## Example 6

In this example, the resulting effective instance has a string value assigned to property called STR. The value of STR is the smallest string value found among all STR properties of instances in the input group. In addition, a numeric property called N is also computed.

```
LVS REDUCE ... [
    effective N
    effective string STR
    N = sum( N )
    STR = min( STR )
]
```

## Effective Property Computation Limitations

Property names used in the EFFECTIVE statement must be simple names; property-name/SPICE-parameter combinations, such as “instpar(w)”, are not accepted in effective property programs. (These are used in Pyxis Layout designs.) Therefore, if a property string similar to instpar(w) is present on devices in the design and an effective property program is provided for that device type, effective properties cannot be calculated during reduction.

## Reduction Program Semantic Checking

---

**Note**

This discussion of reduction program semantic checking is provided for a limited number of users who may be interested in it. For most users, this information may not be needed.

---

The rule file compiler performs semantic checking for properties referenced in device reduction programs. Recall that device reduction programs may appear in the family of (generic or built-in) LVS device reduction specification statements. This semantic checking involves ensuring that device reduction programs are consistent, when applicable, to the [Source System, Layout System](#), and rule file [Device](#) definitions.

To explain this semantic checking, you need to know the concept of *corresponding device definitions*. Given a generic or built-in LVS device reduction specification statement, there is a set of rule file device definitions which are said to correspond to it—that is, definition with computed properties which may fall under the scope of the device reduction specification statement.

- Given a generic LVS Reduce specification statement of the form:
  - LVS Reduce X(A) (explicit component subtype A)
 the set of all corresponding device definitions is as follows:
  - Device X(A) (explicit model name A)
- Given a generic LVS Reduce specification statement of the form:
  - LVS Reduce X (no component subtype)

the set of all corresponding device definitions is as follows:

- Device X (no model name)
- Device X(B) (explicit model name)

where, in the second case, X and B are not the component type and component subtype of any other generic LVS Reduce specification statement.

- Given an [LVS Reduce Parallel MOS](#), [LVS Reduce Series MOS](#), or [LVS Reduce Split Gates](#) specification statement, the set of corresponding device definitions is as follows:

All Device operations with element name MN, MP, ME, MD, M, LDDN, LDDP, LDDE, LDDD, LDD, or any equivalent element names indicated with the [LVS Device Type](#) specification statement.

- Given an [LVS Reduce Parallel Bipolar](#) specification statement, the set of corresponding device definitions is as follows:

All Device operations with element name Q or any equivalent element names indicated with the LVS Device Type specification statement.

- Given an [LVS Reduce Parallel Capacitors](#) or [LVS Reduce Series Capacitors](#) specification statement, the set of corresponding device definitions is as follows:

All Device operations with element name C or any equivalent element names indicated with the LVS Device Type specification statement.

- Given an [LVS Reduce Parallel Resistors](#) or [LVS Reduce Series Resistors](#) specification statement, the set of corresponding device definitions is as follows:

All Device operations with element name R or any equivalent element names indicated with the LVS Device Type specification statement.

- Given a LVS Reduce Parallel Diodes specification statement, the set of corresponding device definitions is as follows:

All Device operations with element name D or any equivalent element names indicated with the LVS Device Type specification statement.

*Consistency checking* is a compilation check whereby a property specified in a device reduction program will actually be computed with correct type (at runtime) by some corresponding rule file device definition.

Consistency checking against rule file device definitions is performed on device reduction programs in the family of LVS Reduce specification statements that operate on extracted layout devices. In Calibre, if the [Layout System](#) specifies a geometric layout, then consistency checking against rule file Device statements is performed on device reduction programs in all device reduction specification statements; if Layout System is SPICE, then such consistency checking is not performed. In ICVerify, consistency checking is performed unconditionally on device reduction programs in all device reduction specification statements.

Given a device reduction program in a device reduction specification statement, consistency checking, when performed, proceeds as follows:

1. There must be at least one rule file device definition corresponding to the device reduction specification statement, with *corresponding* as defined previously.
2. Property names referenced in the device reduction program must be computed in all rule file device definitions corresponding to the device reduction specification statement.
3. Property names referenced in the device reduction program must be computed with the correct type (numeric or string) in all rule file device definitions corresponding to the device reduction specification statement.

This means that property names referenced in the device reduction program must not be computed with vector type in any corresponding rule file device definitions.

## LVS Property Initialize Built-In Language

The [LVS Property Initialize](#) specification statement requires a user program written in a built-in language. This program initializes device properties to specified values in layout and source. The language has all the features of the built-in languages described in [Table 5-1](#) and [Table 5-2](#).

The details specific to the LVS Property Initialize language are described in the following sections:

Writing an LVS Property Initialize Program .....	<a href="#">1867</a>
LVS Property Initialize Characteristics .....	<a href="#">1868</a>
Data Sources .....	<a href="#">1869</a>
PROPerty Statement .....	<a href="#">1869</a>
DEBUG Statement .....	<a href="#">1870</a>
WARN Statement .....	<a href="#">1870</a>
Property Initialization Functions.....	<a href="#">1871</a>

### Writing an LVS Property Initialize Program

Consider this example:

```

1  LVS PROPerty INITialize MN [
2      PROPerty w
3          // shorthand that avoids multiple INP_NUM_VAL() calls:
4      input_w = INPut_NUMERIC_VALue ( w ) // get w if present in input
5      IF ( IS_MISSING ( input_w ) == 1 ) {
6          w = 0 // replace a missing value with 0
7      } ELSE {
8          w = input_w // it is required to assign to all variables
9                  // declared in the leading PROPERTY statement
10     }
11 ]

```

This example illustrates some of the main features of the language. The line numbers on the left are an aid for discussion and are not part of the example text. The language is case-insensitive.

Line 1 shows the LVS Property Initialize statement itself, which in this example applies to all instances in the input databases with element name MN and no model name. Then comes the required opening square bracket "[" followed by the user program.

Line 2 declares the single property variable 'w'. Variable 'w' must be assigned a value somewhere in the body of the user program. The example shows a technique to avoid overwriting 'w' when its value is present in the input database.

Line 4 assigns to a temporary variable named input\_w, the 'w' value present in the input database. Although there is no performance penalty for doing so, this temporary variable lets you avoid making multiple INP\_NUM\_VAL() function calls. This makes the program more readable.

Line 5 uses the IS\_MISSING() function to determine whether or not the ‘w’ property value exists in the input database. If you choose not to use the local variable input\_w approach, line 5 could be re-written like this:

```
IF ( IS_MISSING ( INP_NUM_VAL ( w ) == 1 ) {
    // your choice of methods
```

Line 6 is executed only when the ‘w’ value was NOT found in the input database (it is “missing”). In this case, ‘w’ is initialized with the constant value 0. Note that ‘w’ is only assigned 0 by this program when a ‘w’ value is NOT found for the current instance in the input database.

Line 7 is the ELSE case of the conditional check on line 5; the ELSE branch is taken when a property value for ‘w’ is found in the input database for the current instance.

Line 8 is executed when a valid property value for ‘w’ is found in the input database. Line 8 retains the ‘w’ value that was found in the input database. For instance, say a ‘w’ property value of 2 is found in the input. When this user program concludes, the instance in question retains a ‘w’ property with value 2. Again, the intent of this example program is to demonstrate how to avoid overwriting a ‘w’ property when a valid value is present in the input database for the current instance.

Line 10 shows the concluding brace “}” for the ELSE clause of the conditional IF that began on line 5.

Line 11 shows the concluding square bracket “[”, which is required to close the user program.

## LVS Property Initialize Characteristics

The following table shows information specific to the LVS property Initialize language, which is in addition to the information that appears in [Table 5-1](#).

**Table 5-12. LVS Property Initialize Language Specifics**

Element	Description
Structure	<p>The built-in language has a required sequence of statements, shown here:</p> <pre>[     PROProperty Statement // required, and must be first     DEBUG Statement // optional, must be after Property     // statement and before WARN statement     WARN Statement // optional, must be after optional     // DEBUG statement     property computations a // the properties declared in the     // PROProperty declaration must be     // defined in the program ]</pre> <p>Refer to “<a href="#">Structure</a>” in <a href="#">Table 5-1</a> for general notes on the structure of the language.</p>

**Table 5-12. LVS Property Initialize Language Specifics (cont.)**

Input Instance	The term <i>input instance</i> refers to a layout or source device or instance being operated upon by the LVS Property Initialize program. Property values declared and assigned values in the program are added to instances as they are read from the input databases.
Data Types	The LVS Property Initialize language handles only numeric properties. All numeric literal values are specified in double-precision floating-point. Calculations specified by the user program are also carried out in double-precision. Note that values generated for the input instance are provided to the LVS comparison phase in single-precision floating point.

## Data Sources

Data used in the computation may come from any of the following sources:

- Input database property values — Property values present in the input databases are loaded and made available to user programs when the property names in the input match names of variables declared in the PROPERTY statement. A user program must begin with a PROPERTY statement, which is used to declare property variables to be used in the program.  
  
Note that some property values may not be present in the input databases; for such property values a special “missing” value is made available for the user program to detect. See the definition of the IS\_MISSING() function for a useful way to test for the presence of such “missing” values.
- Numeric Constants — Numeric floating point constants may be used directly within the program.
- Local Variables (Temporaries) — Local variables may be declared and assigned scalar values. Names of local variables must not conflict with property names present in the PROProperty statement.

## PROProperty Statement

The PROProperty statement is used to declare the names of the *generated* numeric properties for which values should be written onto the input instance. The PROProperty statement precedes the optional DEBUG and WARN statements, if present. The PROProperty statement is required. The statement has the following form:

**PROProperty *property\_name* [ ... , *property\_name* ]**

Property names declared in the PROProperty statement must be simple names; property-name/SPIICE-parameter combinations such as instpar(w) are not accepted. To handle such combinations use the [LVS Property Map](#) specification statement.

You should choose property names that are SPICE compliant. See “[General SPICE Syntax](#)” in the *Calibre Verification User’s Manual* for additional information. The rule file compiler will allow you to specify property names that are not SPICE compliant, so care should be taken in choosing names.

## DEBUG Statement

The DEBUG statement is identical to the one discussed under “[DEBUG Statement](#)” on page 1858. It is optional, and if specified, it must appear after the PROProperty statement and before a WARN statement.

## WARN Statement

This optional statement prints a warning to the transcript if the effective property program attempts to perform an erroneous mathematical computation, such as dividing by zero or taking the square root of a negative number. LVS does not generate an error when performing an impossible mathematical computation.

If specified, it must appear after the optional DEBUG statement. The WARN statement has the following syntax and it takes no arguments:

```
WARN
```

If you specify the WARN statement and an impossible mathematical operation is attempted, a message similar to the following in the transcript:

```
*****
LVS PROPERTY INITIALIZE user-program WARNING:
Division by zero (result set to UNKNOWN).
while tracing with:
LVS PROPERTY INITIALIZE ... (within PROPERTY program beginning on rule
file line 27)
Use layout instance ID# 147 in DEBUG trace range.
*****
```

Instance ID 147 can be used in a DEBUG trace range to see exactly how the unknown value was determined for this input instance.

There are no other functions in this language than the ones previously discussed.

## Property Initialization Functions

The following are the built-in functions for the LVS Property Initialize language.

### input\_numeric\_value

This function allows user programs to access, by name, the values of properties present in the input database. The syntax is:

**INPut\_NUMeric\_VALue(*property\_name*)**

The required *property\_name* argument must be a variable declared in the PROPERTY statement. The return value of this function is the related property value present for the current instance in the input database.

If no such property value exists in the input database for the current instance, this function returns the “missing” value instead. The “missing” value is defined internally within the language and may not be accessed directly by user programs; however, presence of the “missing” value may be detected by use of the IS\_MISSING() function.

### is\_missing

This function allows user programs to avoid overwriting property values that are actually present in input data. The syntax is:

**IS\_MISSING(*numeric\_expression*)**

This function requires one *numeric\_expression* argument. Numeric expressions are defined in [Table 5-1](#). If the value of the given *numeric\_expression* is the special “missing” value from an INPut\_NUMeric\_VALue function, then IS\_MISSING( ) returns the numeric constant 1, otherwise the function returns the numeric constant 0.

# Trace Property Computation and Reporting Language

A Trace Property computation and reporting section may be specified in [Trace Property](#) statements. It has all the features of the built-in languages described in [Table 5-1](#) and [Table 5-2](#).

This optional section of a Trace Property statement consists of a user-provided procedure, which defines the operations that are to take place when values of properties on layout instances are compared to values of corresponding properties on source instances. The user-provided procedure lists the names and types of the properties to be made available, and specifies the method of computation and discrepancy reporting to be used with available instance data.

The presence of a Trace Property computation and reporting section attached to a particular Trace Property statement supersedes all other Trace Property statements for the particular device type-subtype pair. You cannot provide any other Trace Property statements for the same device type-subtype pair. When you provide a property computation section, you must declare and handle all properties of interest in the procedure you provide.

The Trace Property language is described in the following sections:

Writing a Trace Property Program .....	1872
Trace Property Language Characteristics .....	1875
Data Sources .....	1876
PROPerty and PROPerty STRING Statements .....	1877
DEBUG Statement .....	1878
WARN Statement .....	1879
Trace Property Function Summary .....	1879
Trace Property Function Reference .....	1880
Trace Property User Computation Example .....	1883
Trace Property Program Limitations .....	1884

## Writing a Trace Property Program

Consider this example:

```

1  TRACE PROPERTY R [
2      PROPERTY r
3      //
4      tolerance = 0.001 // "constant" stored as a local variable
5      discrepancy = 0
6      //
7      lay_r = LAYout_NUMeric_VALue(r) // retrieve r from layout
8      src_r = SOUrce_NUMeric_VALue(r) // retrieve r from source
9      //
10     if ( src_r != 0 ) { // prefer src value as denominator
11         diff = ABS( lay_r-src_r )
12         discrepancy = 100 * diff / src_r
13     } else if ( lay_r != 0 ) { // lay value as denominator
14         diff = ABS( lay_r-src_r )
15         discrepancy = 100 * diff / lay_r
16     } else {

```

```

17         REPort_MESSAGE( "A discrepancy exists; value cannot be
calculated due to div. by zero")
18     }
19     //
20     if ( discrepancy != 0 && discrepancy > tolerance ) {
21         REPort_NUMERIC_DISCrepency ( r , discrepancy )
22     }
23 ]

```

This example illustrates many of the features of the language. The line numbers on the left are an aid for discussion and are not part of the language. The language is case-insensitive.

In the following section, *input pair* means a layout instance and its single, corresponding, source instance. Property values are available based upon this input pair.

Line 1 shows the standard Trace Property statement with its property computation section. The literal square brackets are required to indicate the presence of the Trace Property computation section.

Line 2 contains the PROProperty declaration statement. This statement declares the names of all properties for which layout values and source values should be made available throughout the rest of the procedure. Each property listed here may be used in data retrieval functions shown later (see lines 7 and 8). The names here represent both layout values (properties found on the layout instance) and source values (properties found on the corresponding source instance).

The [LVS Property Map](#) specification statement applies to names in the PROProperty declaration statement, and is used to derive layout-specific and source-specific property names based on the *simple name* given in LVS Property Map statement and used by the Trace Property computation program.

Lines 3, 6, 9, and 19 show a comment character.

Line 4 declares a local variable, tolerance, and assigns it the value 0.001. This is used as a symbolic constant later in the procedure.

Line 5 declares a local variable, discrepancy, and initializes it to 0. It is used later in the procedure to store a discrepancy value for possible reporting.

Line 7 retrieves the value of a property named 'r' from the layout instance, while line 8 retrieves r's value from the corresponding source instance.

Lines 10-18 show a method for avoiding use of zero values from the input pair. If you provided the statement:

```
REPort_NUMERIC_DISCrepency(r,100*ABS((lay_r-src_r)/src_r))
```

and the 'r' value on the source instance was zero, a discrepancy would be reported as the Unknown value (represented by '?' in the LVS Report file). The Unknown value is

assigned because division by zero is attempted and is undefined. Instead, lines 10-18 take this approach:

Line 10 checks to see if the source instance's r value is non-zero. If so, lines 11-12 calculate the discrepancy value using the non-zero src\_r as the denominator. The result is stored in the local variable 'discrepancy'.

Line 13 checks to see if the layout instance's r value is non-zero. If so, lines 14-15 calculate the discrepancy value using the non-zero lay\_r as the denominator. The result is stored in the local variable 'discrepancy'.

Line 16 is a fallback in case both lay\_r and src\_r are zero. In this case, no discrepancy value can be calculated. Instead, line 17 reports a discrepancy as a literal string message, which appears in the LVS Report file.

Lines 10-18 also show how braces { } can be used to group multiple statements into a single logical statement. As in C or C++, this grouping is useful for indicating the extent of a dependent clause for a conditional statement.

Note that you could replace lines 11-12 with the single statement:

```
discrepancy = ABS( (lay_r-src_r)/src_r )
```

Line 20 checks whether the discrepancy value is non-zero, and whether it exceeds a tolerance value. If so, line 21 reports a numeric discrepancy for property r and the input pair of instances. The source value and layout values of r appear in the LVS Report file, along with the 'discrepancy' value calculated in the preceding procedure.

## Trace Property Language Characteristics

The following table shows information specific to the Trace Property computation language, which is in addition to the information that appears in [Table 5-1](#).

**Table 5-13. Trace Property Language Specifics**

Element	Description
Structure	<p>The built-in language has a required sequence of statements, shown here:</p> <pre>[ // see <a href="#">PROProperty and PROProperty STRING Statements</a> // at least one of these statements must appear first in the // program. Both may be present, and they can be in any order. Property Statement Property String Statement <a href="#">DEBUG Statement</a> // optional, must be after Property // statement and before <a href="#">WARN Statement</a> <a href="#">WARN Statement</a> // optional, must be after optional // DEBUG statement property computations // the properties declared in the // PROProperty declaration must be // defined in the program ]</pre> <p>Refer to <a href="#">Structure</a> in <a href="#">Table 5-1</a> for general notes on the structure of the language.</p>
Input Pair	Refers to a layout instance and its single corresponding source instance. Property values are made available for the user-provided procedure for retrieval by name from layout and source based upon this input pair.
Data Types	The property computation language handles both numeric and character-string properties. All numeric literal values are specified in double-precision floating-point. Note that values read from the input pair are provided by the LVS comparison phase in single-precision floating-point; they are converted to double-precision floating-point for use by the user-provided procedure. Calculations specified by the user procedure are also carried out in double-precision. Discrepancy results, if any, are returned to the LVS comparison phase in single-precision floating-point.
String Expressions	The only string expressions that are supported are literal string constants enclosed in double quotes (""), and string values returned by the LAyout_STRING_VALUE() and SOURCE_STRING_VALUE() data retrieval functions. These may be used only in the context of the REPort_MESSAGE() and STRING_COMPARE() functions.

**Table 5-13. Trace Property Language Specifics (cont.)**

Logical Expressions	String expressions may not directly participate in logical expressions. Note, however, the definition of the <a href="#">string_compare</a> function.
---------------------	---

## Data Sources

Data used in the computation and discrepancy reporting may come from any of the following sources:

- Numeric Constants — Numeric floating-point constants may be used directly within the procedure.
- Instance Data — Property data associated with instances in the input pair may be accessed by name using built-in data retrieval functions. Examples are the LAYout\_NUMERIC\_VALue() and SRC\_STRING\_VALue() functions.

When a property name “X”, declared in a PROPerTY or PROPerTY STRING statement, is used inside a data retrieval function, it refers to the layout or source property value bound to an instance in the input pair. The layout- or source-specific value retrieved is based on the name of the data retrieval function used. (See definitions of LAYout\_NUMERIC\_VAL(*prop\_name*) and similar functions under [Trace Property Function Reference](#).)

It is not valid to read “X” directly without the use of a data retrieval function, nor to assign to “X”. The use of “X” in such cases would be ambiguous with respect to the [layout,source] input pair. For example, the following causes compiler errors:

```
TRACE PROPERTY t(st) [
    PROPERTY q
    q = 0           // ERROR. Ambiguous. Does 'q' mean
                    // layout or source q value?
                    // Is 'q' meant to be a local?
    local = q + 1   // ERROR. Which 'q' -- layout or source?
    local = lay_val(q) + 1 // OK.
]
```

The first error above illustrates that assignment to a PROPerTY name is invalid. There is no notion of writing property values when performing a Trace Property comparison and reporting procedure.

The second error above illustrates that PROPerTY names cannot be used in generic expressions, nor is this a good idea, as q alone is ambiguous given the presence of both a layout and source value for q.

These internal restrictions make potentially confusing code invalid, as shown here:

```
TRACE PROPERTY MP [
    PROPERTY pd
    if ( lay_val(pd) != src_val(pd) ) {
        pd = 0 // confusing; is this intended to be a local?
        //
        rep_disc ( pd, pd )
        // ^^ First parameter must be a property name.
        // Second parameter must be an expr, but is not in this
        // case.
    }
]
```

- Local Variables (Temporaries) — Local variables may be declared and assigned scalar values. Names of local variables must not conflict with property names present in the PROProperty or PROProperty STRING statement. Only numeric local variables are supported. Assignment to string local variables is not supported.

## PROProperty and PROProperty STRING Statements

The PROProperty or the PROProperty STRING statements are used to declare the names of the properties for which values should be made available from layout and source input pair to the computation. The PROProperty statement declares numeric properties and the PROProperty STRING statement declares string properties (that is, properties with character-string values). Either one or the other is required, and both may appear. The PROProperty statement may precede or follow the PROProperty STRING statement, but they must both precede the optional DEBUG and WARN statements, if present. These statements have the following form:

```
PROProperty property_name [ ... , property_name ]
PROProperty STRING property_name [ ... , property_name ]
```

Properties listed here are considered read-only. They may only be accessed by data retrieval and discrepancy reporting functions defined later.

Property names declared in PROProperty or PROProperty STRING statements must be simple names; property-name/SPICE-parameter combinations such as instpar(w) are not accepted (these are typically found in Pyxis Layout designs). To handle such combinations, use the [LVS Property Map](#) specification statement.

In the following example, four numeric properties are declared with names W, L, AS, and AD. Also, two string properties are declared with names W\_STR and L\_STR.

```
PROPERTY W, L, AS, AD
PROPERTY STRING W_STR, L_STR
```

A property name cannot be declared in both an PROProperty and PROProperty STRING statement within the same property computation procedure. For example, the following causes a compiler error:

```
TRACE PROPERTY ... [
    PROPERTY x
    PROPERTY STRING x
```

```
// ERROR. 'x' already mentioned in PROPERTY
IF ( LAY_NUM_VAL(x) == 0 )
    EP_MSG ( "unexpected zero value for 'x'" )
]
```

Again, names mentioned in PROProperty and PROProperty STRING declaration statements may only be used in data retrieval and discrepancy reporting functions.

Note that property names must be declared in order to be used in data retrieval or discrepancy reporting functions. For example, the following cause compiler errors:

```
TRACE PROPERTY t(st) [
    PROPERTY q
    rpt_disc(w,0.1)
    // ERROR. 'w' is not a declared PROPERTY name
    local = lay_num_val(l)
    // ERROR. 'l' is not a declared PROPERTY name
]
```

Note that properties mentioned in Trace Property computation PROProperty or PROProperty STRING declaration statements are considered *actionable* properties, because they are declared for the purposes of trace comparison by a user procedure. As actionable properties, they trigger missing property errors when such properties are missing from a device in either instance of the input pair. Other actionable properties include those mentioned in TOLERANCE statements (within Device reduction programs), LVS Filter, and LVS Split Gate Ratio specification statements.

## DEBUG Statement

Monitoring the execution of Trace Property computation procedures can be useful in finding errors during the development of new Trace Property computation code. Monitoring is controlled by the DEBUG statement, which is optional. When present, DEBUG must follow the PROProperty and PROProperty STRING statements, and precede the optional WARN statement. It has the following form:

**DEBUG range1 [ , range2 , ... rangeK ]**

where each range is either an instance ID, or a pair of instance IDs separated by a hyphen “-”. The instance IDs represent both the layout and source device in the input pair. Instance IDs are unique across all devices in the design. Instance IDs are assigned by the system and are not guaranteed to be sequential, merely unique. These IDs are most useful in debugging warnings produced when the WARN statement is present, and an impossible mathematical operation is attempted.

Here is an example of a DEBUG statement.

**DEBUG 0-2, 508**

This statement causes a step-by-step report of the Trace Property computation and discrepancy reporting activities to be printed for each of the device instances 0, 1, 2, and 508. At least one range must be specified, although it can contain only a single instance. The monitoring output can be voluminous, so the use of small ranges is recommended. Use of the DEBUG statement is

not recommended in production rule files for the same reason. Again, the instance numbers can be obtained from output produced by the **WARN** statement.

## **WARN Statement**

If any “impossible” mathematical computations are performed (such as division by zero), the result of the computation is set to the unknown value. If such a value is reported as a discrepancy, it appears as ‘?’ in the LVS Report file.

Note that attempting an impossible mathematical computation is not considered an error that stops LVS comparison. If you want to be notified of such attempted computations, use the **WARN** statement. This statement takes no arguments:

```
WARN
```

If you specify the **WARN** statement, then it must appear in the Trace Property computation procedure after the **PROPerty**, **PROPerty STRING** and optional **DEBUG** statements. If you specify the **WARN** statement, then, when an operation such as division by zero or square root of a negative number is attempted, you see a message similar to the following in the transcript:

```
*****
TRACE PROPERTY user-program WARNING:
Division by zero (result set to UNKNOWN).
while tracing with:
TRACE PROPERTY ... (within PROPERTY program beginning on rule file line
27)
Use layout instance ID# 147 in DEBUG trace range.
Use source instance ID# 4 in DEBUG trace range.
*****
```

Instance IDs 147 or 4 can be used in a **DEBUG** trace range to see exactly how the unknown value was determined for this input group.

## **Trace Property Function Summary**

[Table 5-14](#) summarizes the functions available in the Trace Property language.

**Table 5-14. Summary of Trace Property Language Functions**

Function	Definition
Data (Property Value) Retrieval Functions	
<a href="#">layout_numeric_value</a>	Returns a layout numeric property.
<a href="#">source_numeric_value</a>	Returns a source numeric property.
<a href="#">layout_string_value</a>	Returns a layout numeric property.
<a href="#">source_string_value</a>	Returns a source numeric property.
String-Related Functions	
<a href="#">string_compare</a>	Compares two strings.

**Table 5-14. Summary of Trace Property Language Functions**

Function	Definition
Discrepancy Reporting Functions	
<code>report_numeric_discrepancy</code>	Writes a numeric discrepancy (ERROR percentage) to the LVS Report file.
<code>report_numeric_value</code>	Writes a numeric discrepancy (ERROR absolute) to the LVS Report file.
<code>report_string_discrepancy</code>	Writes a string inequality discrepancy to the LVS Report file.
<code>report_message</code>	Causes a Trace Property discrepancy and writes message string to the LVS Report file.

## Trace Property Function Reference

The strings shown in capital letters are acceptable shortened forms of the function names.

### **layout\_numeric\_value**

#### **LAYOut\_NUMerIc\_VALue(*prop\_name*)**

Returns the value of numeric property *prop\_name* from the layout instance.

The *prop\_name* must be a property declared in a PROPerty declaration statement. It is a compiler error to use a string property for *prop\_name*.

A numeric value is expected for the property being read. If a numeric value is not found, the “missing” value is used in place of the original value.

### **layout\_string\_value**

#### **LAYOut\_STRInG\_VALue(*prop\_name*)**

Returns the value of string property *prop\_name* from the layout instance.

The *prop\_name* must be a property declared in a PROPerty STRING declaration statement. It is a compiler error to use a numeric property for *prop\_name*.

A string value is expected for the property being read. If a string value is not found, the “missing” value is used in place of the original value.

### **source\_numeric\_value**

#### **SOURCE\_NUMerIc\_VALue(*prop\_name*)**

Returns the value of numeric property *prop\_name* from the source instance.

The *prop\_name* must be a property declared in a PROPerty declaration statement. It is a compiler error to use a string property for *prop\_name*.

A numeric value is expected for the property being read. If a numeric value is not found, the “missing” value is used in place of the original value.

## source\_string\_value

### SOURCE\_STRING\_VALUE(*prop\_name*)

Returns the value of string property *prop\_name* from the source instance.

The *prop\_name* must be a property declared in a PROProperty STRING declaration statement. It is a compiler error to use a numeric property for *prop\_name*.

A string value is expected for the property being read. If a string value is not found, the “missing” value is used in place of the original value.

## report\_message

### REPort\_MESSAGE(*literal\_string\_constant*)

Causes a Trace Property discrepancy, and writes *literal\_string\_constant* (a character string enclosed in double quotes) to the LVS Report file.

## report\_numeric\_discrepancy

### REPort\_NUMeric\_DISCrepency(*prop\_name*, *numexpr*)

Writes a numeric discrepancy for *prop\_name* to the LVS Report file as a percentage difference. The *numexpr* may be any numeric expression supported by the language. This function does not calculate a percentage. It simply reports *numexpr* as a percentage.

Here is a report example:

```
*****
          PROPERTY ERRORS
DISC#   LAYOUT           SOURCE           ERROR
*****
1      M4(13.375,14.000)  MN(N)
nx: 6                                     M4  MN(N)
                                         nx: 4          50%
```

## report\_numeric\_value

### REPort\_NUMeric\_VALue(*prop\_name*, *numexpr*)

Writes a numeric discrepancy for *prop\_name* to the LVS Report file as an absolute difference. The *numexpr* may be any numeric expression supported by the language.

Here is a report example:

```
***** PROPERTY ERRORS *****
DISC# LAYOUT SOURCE ERROR
*****  
1 M4(13.375,14.000) MN(N)
nx: 6 M4 MN(N)
nx: 4 2
```

## report\_string\_discrepancy

**REPort\_STRING\_DISCrepency(*prop\_name*)**

Writes a string inequality discrepancy for *prop\_name* to the LVS Report file. The *prop\_name* must represent a string-valued property.

## string\_compare

**STRING\_COMPare(*strexpr1*, *strexpr2*)**

C.compares the input strings *strexpr1* and *strexpr2* and returns the values given in [Table 5-15](#).

**Table 5-15. STRING\_COMPare Return Values**

Return Value	Condition
0	values of <i>strexpr1</i> and <i>strexpr2</i> are equivalent
-1	<i>strexpr1</i> is lexicographically less than <i>strexpr2</i>
1	<i>strexpr1</i> is lexicographically greater than <i>strexpr2</i>

The parameters *strexpr1* and *strexpr2* are either literal string constants enclosed in double quotes (" ") or string-type property values returned by `lay_str_val()` or `src_str_val()`. Local string variables are not supported and cannot be given as arguments to `string_compare()`.

**Case sensitivity** — Controlled by the current setting of the [LVS Compare Case](#) specification statement.

Note that `string_compare()` behaves like the C `strcmp()` routine, except for the case-handling behavior. This function is intentionally abbreviated as `STR_COMP()` (including variations) and not `strcmp()`, so as to differentiate its case-handling behavior from that of the familiar C `strcmp()` routine.

## Trace Property User Computation Example

The following example examples of most of the commands in the language. Assume property r is 2.0 in the layout and 3.0 in the source. Assume property str is “string1” in the layout and “string2” in the source.

```

TRACE PROPERTY R [
    PROPERTY r
    PROPERTY STRING str

    // in this program, tolerance is handled as a percentage.
    tolerance = 0.1 // "constant" stored as a local variable.
    discrepancy = 0

    lay_r = layout_numeric_value(r) // retrieve r value from layout
    src_r = source_numeric_value(r) // retrieve r value from source

    if ( src_r != 0 ) { // prefer src value as denominator
        diff = ABS( lay_r-src_r )
        discrepancy = 100 * diff / src_r
    } else if ( lay_r != 0 ) { // lay value as denominator
        diff = ABS( lay_r-src_r )
        discrepancy = 100 * diff / lay_r
    }
    // if discrepancy > tolerance, report it.
    // the discrepancy is reported as a percentage.
    if ( discrepancy > tolerance ) {
        report_numeric_discrepancy( r , discrepancy )
    }
    // compare string property values
    if ( string_compare(layout_string_value(str),source_string_value(str))
!= 0 ) {
        report_string_discrepancy(str)
    }
    // The following is just an example to demonstrate message
    // discrepancy reporting.

    report_message( "This discrepancy is always reported" )
]

```

This routine produces results similar to these in the LVS Report file:

```
*****
PROPERTY ERRORS

DISC#  LAYOUT                      SOURCE                      ERROR
*****
1      R0(x,y)  R(x)                  R2  R(x)
      l: 0.4 u                         l: 0.6 u                 33%
      w: 2 u                           w: 3 u                 33%

2      R0(x,y)  R(x)                  R2  (x)
      str: string1                    str: string2

3      R0(x,y)  R(x)                  R2  (x)
      ERROR:  "This discrepancy is always reported"
```

## Trace Property Program Limitations

The following sections document limitations with Trace Property programs.

### Limitations On Use of Strings and String Properties

This section summarizes the various limitations involving string properties in the Trace Property computation and reporting language. Most of these limitations have been mentioned previously.

- String properties are supported in limited contexts. Specifically: string properties may only appear as the single argument to a string-valued data retrieval function (`lay_str_val()` or `src_str_val()`), as one of the parameters to the `str_comp()` function, or as the single parameter to the `rep_str_disc()` function.
- Properties declared as PROProperty STRING may not appear in any numeric context. The following example generates a compiler error:

```
...
PROPERTY STRING str
x = LAY_NUM_VAL(str) // Error. Trying to retrieve numeric.
...
```

- The Trace Property computation and reporting language does not support string-valued local variables. Assignment of literal string values is not supported. Assignment of string-valued variables to another variable is not supported. Use of any of these forms results in a compiler error. For example:

```
...
PROPerty STRING str
str = "my_string" // Error, no string literals in assignment
s1 = SRC_STR_VAL(str)
// Error, no local variable string assignment.
s1 = str      // Error, no variable to variable string assignment.
...
```

### Missing and Unknown Values Present on the Input Pair

The following actions, listed in highest-to-lowest-priority order, are taken when Missing or Unknown property values are present on instances in the Input Pair. Missing and Unknown values are defined previously.

1. If any Unknown property values are observed on either instance in the input pair, a Trace Property discrepancy is recorded in the LVS Report against each related property name. This is done for all observed Unknown property values. If the `WARN` statement is present in the user program, a warning is printed to the transcript to describe what happened. The body of the user program is not executed.
2. If all property values on the source instance of the input pair are missing, or all property values on the layout instance of the input pair are missing, the body of user program is not executed. No Trace Property discrepancies are recorded in the LVS Report. Instead,

missing property errors are recorded to the LVS Report according to the setting of LVS Report Option E. If the WARN statement is present in the user program, a warning is printed to the transcript to describe what happened.

3. If some of the property values on the source instance of the input pair are missing (the other source property values are present) or some of the property values on the layout instance of the input pair are missing (the other layout property values are present), then the user program is run. If WARN is present in the user program, a warning is printed to the transcript to describe what happened.

When case 3 occurs (the user program is executed with some missing values present) flow control and evaluation of logical expressions are affected as follows:

- If a Missing value is presented to the equality operator (==), the result of the comparison is false. That is, a Missing value is always considered to fail the equality test. In the following example, the false (ELSE) branch of the IF...ELSE statement will always be executed:

```
TRACE PROPERTY c [
    PROPERTY c
    zero = 0
    local = lay_num_val(c)/zero
    IF(local == 0) {
        REP_MSG("true branch")
    } ELSE {
        REP_MSG("false branch") // this branch is executed
    }
]
```

- If a Missing value is presented to the inequality operator (!=), the result of the comparison is true. That is, a Missing value is always considered to pass the inequality test. In the following example, the true (IF) branch of the IF...ELSE statement will always be executed:

```
TRACE PROPERTY c [
    PROPERTY c
    zero = 0
    local = lay_num_val(c)/zero
    IF(local != 0) {
        REP_MSG("true branch") // this branch is executed
    } ELSE {
        REP_MSG("false branch")
    }
]
```

- If a Missing value is presented to any of the other relational operators (<, <=, >=, >) the comparison is always false. That is, a Missing value always considered to fail any of the following relational tests: less-than, less-than-or-equal-to, greater-than-or-equal-to, greater-than. In the following example, the false (ELSE) branch of the IF...ELSE statement is ARBITRARILY chosen for execution:

```
TRACE PROPERTY c [
    PROPERTY c
```

```
zero = 0
local = lay_num_val(c)/zero
IF(local > 0) {
    REP_MSG("true branch never executed")
} ELSE {
    REP_MSG("false branch executed arbitrarily")
    // this branch executed
}
]
```

When writing a trace property computation procedure, you may wish to check whether a property value is either missing or present. The following example shows a method of performing this check:

In this example, assume that the ‘c’ property value is missing in the layout, but is present in the source.

```
1 TRACE PROPERTY c [
2     PROPERTY c
3     lc = LAY_NUM_VAL(c)
4     sc = SRC_NUM_VAL(c)
5     if ( !(lc < 0) && !(lc >= 0) ) {           // layout c is missing
6         REP_MSG ( "c missing in layout" )
7     } else if ( !(sc < 0) && !(sc >= 0) ) { // source c is missing
8         REP_MSG ( "c missing in source" )
9     } else {                                // both values are present
10        // do real comparison procedures
11    }
12 ]
```

When ‘c’ is missing in the layout, this example will report “c missing in layout” as a trace property discrepancy in the LVS report. The LVS comparison result will be INCORRECT.

Line 5 can be alternatively written (using DeMorgan’s Theorem) as:

```
if ( !((lc < 0) || (lc >= 0)) ) ...           // layout c is present
```

## Trace Property Programs May Affect LVS Performance

If you use a Trace Property program to replace built-in Trace Property behavior, performance of the LVS comparison phase may be degraded. In one design, mimicking built-in Trace Property behavior with an equivalent program resulted in the overall execution time for LVS growing by a factor of 1.6×. The LVS cumulative comparison execution time grew by a factor of 3.2×. Most of this additional time was spent in the Ambiguity Resolution phase, where the execution time grew by a factor of 3.9×.

## Device Statement Property Consistency Checking Not Performed

Recall that the built-in Trace Property statements have *consistency checking* performed for properties named in the Trace Property statement. Briefly, consistency checking is a compilation check that a property specified in a mask mode Trace Property statement will actually be computed with correct type (at runtime) by some corresponding rule file Device definition.

Such consistency checking of property names used in Trace Property programs and LVS Property Initialize programs is not performed against corresponding Device definitions at rule file compilation time.



# Chapter 6

## Tcl Verification Format

---

### Introduction

This document describes the Tcl Verification Format (TVF) programmable extension to the Standard Verification Rule Format (SVRF) language for Calibre. TVF is employed in two distinct environments: compile-time and runtime.

**Compile-Time TVF** is Tcl code that is interpreted at compilation time when you initiate a Calibre job. Calibre reads the compile-time TVF script and translates it directly into SVRF commands at the start of the run. Compile-time TVF contains no elements that interact with runtime data.

**Runtime TVF** is Tcl code that appears within a special SVRF element called TVF Function. Runtime TVF allows you to access data generated during a Calibre run. It can be used with the **TVF** layer operation to access layer data, perform layer operation, and output layer data.

There are similarities and differences between compile-time and runtime TVF. Among the important differences to bear in mind are these:

- Compile-time TVF code appears in a compile-time script beginning with “#!tvf” as the opening line. It cannot appear in a typical SVRF rule file. Runtime TVF appears within an SVRF element called TVF Function. Runtime TVF can appear either in a typical SVRF rule file, or within a compile-time TVF script.
- Compile-time TVF is translated into SVRF commands at the beginning of a run, which are then fed to the SVRF compiler. Runtime TVF appears within an SVRF function, and each runtime TVF script operates in its own environment during the run.

These details are discussed later in this chapter.

TVF uses all standard Tcl constructs, which provide a uniform programming structure. A Tcl script consists of a sequence of command lines executed whenever the parser encounters a command. The command name is the first word of the command, followed by the arguments passed to the command.

It is assumed that you are already familiar with SVRF and Tcl programming. The “[Key Concepts](#)” chapter discusses important aspects of the SVRF language.

### TVF License Usage

When using compile-time or runtime TVF in conjunction with nmDRC or nmLVS, Calibre requires no additional licenses. If you are preprocessing a TVF file using the calibre -E option (see “[Compile-Time TVF File Usage](#)”), the appropriate licenses must be available for the statements used in your TVF file.

## Differences Between TVF and SVRF

Before using TVF, you should be aware of some fundamental differences between Tcl and SVRF. Tcl and SVRF have conflicting interpretations of certain language features including several command names, character usage, whitespace, case-sensitivity, and so forth. Table 6-1 shows most of the major differences between runtime TVF and SVRF.

**Table 6-1. Tcl/TVF Versus SVRF Syntax**

Language element	Tcl/TVF interpretation	SVRF interpretation
Case-sensitivity	Case-sensitive.	Case-insensitive.
Comment characters (#, //, @, /* */)	Uses # character at the beginning of a line. TVF has additional comment commands.	Uses C-style comment characters // and /* */. Uses @ symbol for rule check user comments.
Dollar sign (\$)	Dereferences variables.	Dereferences environment variables in pathnames, cell names, and preprocessor directives.
Macros	Handled using standard Tcl procs. Does not use SVRF Dmacro and Cmacro statements.	Use Dmacro and Cmacro statements.
Preprocessor directives (#)	TVF uses a Tcl preprocessor. Does not use SVRF-style directives.	Uses #define, #undefine, #ifdef, #ifndef, and #else statements.
Environment variables	Handled using the standard Tcl methods.	Uses the Variable specification statement. Also allows \$variable call for pathnames.
Newline	Newline character terminates a command (usually).	Insensitive to whitespace.
Numeric integer constants with leading 0	Interpreted as octal.	Interpreted as decimal.
Quotation marks (" ", ' )	Double quotes (" ") delimit a substitutable string. Single quotes are not special.	Double quotes and single quotes (' ') delimit a string, as in a cell name, layer name, or rule check name.
Brackets and braces [ ] { }	[ ] cause the interpretation of the contents. { } delimit a list of items.	[ ] delimit parameters. { } delimit a rule check.

**Table 6-1. Tcl/TVF Versus SVRF Syntax (cont.)**

Language element	Tcl/TVF interpretation	SVRF interpretation
Semicolon (:)	Denotes start of a new line.	Invalid.
Double colon (::)	Denotes a namespace.	Vector retrieval operator.

Note that Compile-time TVF allows specification of native SVRF statements within the tvf::VERBATIM function, so in that sense, SVRF statements are directly supported within Compile-time TVF. Not all SVRF statements are shadowed as native TVF commands, however.

In addition to the special characters discussed in [Table 6-1](#), the SVRF language assigns special meanings to these characters:

? = < == > <= >= != - + \* / ^ ! ~ % && || ,

which are not generally used in the same way as in Tcl. See [Table 2-1](#) for the meaning of these characters in the SVRF language.

The following are other differences that need careful consideration:

- SVRF allows multiple statements per line. Tcl does not, unless you use the semicolon (;) to delimit the end of a statement.
- An SVRF statement may span multiple lines. Tcl commands generally do not, unless you use a backslash (\) to indicate line continuation. Be sure there are no spaces or tabs after a line continuation character or an error results.
- The special characters [ ], and \$ must be protected by braces ({} ) or a backslash (\) character to avoid premature evaluation in a compile-time TVF script.
- Square brackets [ ] that occur as a part of SVRF command syntax must be escaped in a runtime TVF script. For example, the SVRF command:

**EXT [layer] < 1.0**

should be written as:

**EXT \[layer\] < 1.0**

when it appears within a runtime TVF script. This is not necessary for SVRF syntax that appears in a tvf::VERBATIM block, as discussed in “[Passing Verbatim SVRF Code in Compile-Time TVF](#)”.

## Compile-Time TVF

Compile-time TVF appears in a special rule file that has a script of compile-time TVF commands. This script is interpreted and translated directly into SVRF code, one line at a time,

upon execution of the Calibre command. Compile-time TVF contains no elements that access data during the Calibre run.

Compile-time TVF cannot be placed into a normal SVRF rule file; however, the TVF script filename can be passed to Calibre just like any other rule file name.

## Compile-Time TVF Commands

This section discusses TVF compile-time commands that are translated into SVRF code and compiled at the beginning of a Calibre job. Compile-time TVF commands allow you to do two particularly useful things:

- Write Tcl procs that can be repeatedly called.
- Write loops that generate multiple SVRF rule checks in a few lines of code.

The main advantage of compile-time TVF over SVRF is you can write your rule file in fewer lines of code using TVF constructs.

## Opening Line

The opening line of a compile-time TVF script, which is loaded like an SVRF rule file, must begin with this string:

```
#!tvf
```

This instructs Calibre to start in programmable mode using the TVF preprocessor for interpreting the script.

## Command Syntax

All TVF compile-time commands are embedded in the tvf:: namespace, which can be thought of as a library of commands with the special prefix “tvf::”.

Tcl commands consist of a single word followed by a list of arguments. The basic syntax for TVF commands is this:

```
tvf::<command_name> <arguments>
```

The first token after the tvf:: is a case-sensitive command name, followed by whitespace, followed by appropriate arguments. For example:

```
tvf::SETLAYER gate = "poly AND diff"  
tvf::OUTLAYER "COPY $one_not_two"  
tvf::COMMENT {This is one type of compile-time comment.}
```

TVF commands have upper- and lowercase versions, with a few exceptions where collisions with Tcl command names would occur.

## Passing Verbatim SVRF Code in Compile-Time TVF

Compile-time TVF provides the following command to pass in verbatim SVRF code:

```
tvf::VERBATIM {  
    <verbatim SVRF code>  
    ...  
}
```

Whatever appears between the braces is passed directly to the SVRF compiler and must adhere to SVRF syntactical conventions. Any errors in the SVRF syntax generate a Calibre compiler error, as usual.

You should use this command as much as possible in your compile-time TVF scripts to pass blocks of SVRF code that do not need to be changed to TVF syntax. This includes runtime TVF commands.

The following example shows how a typical SVRF rule check can be passed as-is within the VERBATIM command.

### Example 6-1. VERBATIM Statement

```
tvf::VERBATIM {  
    //Minimum m1 width is 0.10  
    m1_width {  
        INT m1 < .10 ABUT < 90 SINGULAR  
    }  
}
```

When using [Include](#) statements inside of a tvf::VERBATIM block, the file referenced by the Include must be written in SVRF. When using a tvf::INCLUDE statement, the file referenced must also be in SVRF. If a TVF file is to be referenced, use the Tcl “source” command.

## SVRF Statements Mapped to TVF

Tcl commands are the first word of a Tcl string, with the remainder of the words being passed as arguments to the command. This feature of Tcl determines how SVRF statements are mapped to compile-time TVF commands.

Most SVRF specification statements, device extraction statements, connectivity extraction operations, and fracture operations are easy to translate into compile-time TVF commands. Specification statements occur in families. Each family has a predictable initial keyword. Each initial keyword is mapped to a TVF command using the tvf:: namespace prefix. The remainder of the words and parameters in the SVRF statements are simply passed to the TVF command as arguments.

Layer operations generally do not have this feature. Layer operations must appear either in a layer derivation (like  $x = y \text{ AND } z$ ) or in a rule check block that begins with an arbitrary name.

In such cases, there is no way to predict the first token of these constructs. Hence, there is no way to map the layer operation name as a command in compile-time TVF.

As one exception, the Copy operation is supported as a compile-time TVF command. This is for historical reasons. You generally will not use tvf::COPY in your rule file.

As another exception, the DFM and MDP families of layer operations must begin with a consistent keyword, so they are mapped to compile-time TVF as commands. However, because these layer operations must appear in the context of either a layer derivation or a rule check, there generally is no need to use the TVF commands mapped to them.

The list of SVRF statement families that are mapped to TVF is in [Table 6-2](#):

**Table 6-2. Specification Statement Families Mapped to TVF Commands<sup>1</sup>**

ATTACH	FLATTEN <sup>3</sup>	MDP	RESOLUTION
CAPACITANCE	FRACTURE	MDPMERGE	SCONNECT
CONNECT	GROUP	MDPSTAT	SNAP <sup>7</sup>
COPY	HCELL	MDPVERIFY	SOURCE
DBCLASSIFY	INCLUDE	PARASITIC VARIATION	SVRF
DEVICE	INDUCTANCE WIRE	PERC	TEXT
DFM	LABEL	PEX <sup>5</sup>	TITLE
DISCONNECT	LAYER	POLYGON	TRACE PROPERTY
DRC	LAYOUT	PORT	TVF FUNCTION
ERC	LITHO <sup>4</sup>	PRECISION	UNIT
EXCLUDE	LVS	PUSH <sup>6</sup>	VARIABLE
EXPAND <sup>2</sup>	MASK	RESISTANCE	VIRTUAL
FLAG			

1. These commands all have lowercase versions except SOURCE, TRACE, VARIABLE, and UNIT.

2. The Expand Edge and Expand Text layer operations are not included.

3. The Flatten layer operation is not included.

4. Litho setup files do not use TVF.

5. PEX Inductance statements are not currently included.

6. The Push layer operation is not included.

7. The Snap layer operation is not included.

The first word of an SVRF statement is mapped to a TVF command name (always with the tvf:: namespace prefix). For example, the DRC Check Map SVRF specification statement is mapped in TVF as:

```
tvf::DRC Check Map ...
```

or

```
tvf::drc Check Map ...
```

The TVF command name is tvf::DRC or tvf::drc, with the remainder of the specification statement syntax being passed as arguments to the command, which the SVRF compiler interprets. The remainder of the arguments after the TVF command name are case-insensitive.

Note that tvf::DRC and tvf::drc encompass an entire family of SVRF specification statements that begin with the string “DRC”. As stated previously, *only the first word of a family of SVRF statements is used as a TVF command name*. The remainder of the SVRF statement arguments are simply passed to the SVRF compiler for processing.

**SVRF statements with special text case handling**—To avoid command name collisions, the specification statements beginning with the words:

- SOURCE
- TRACE
- VARIABLE
- UNIT

are not mapped in lowercase within TVF. Tcl has lowercase commands with these names, so to avoid a collision of command names, TVF has only uppercase versions of these commands.

### Example 6-2. SVRF to TVF Translation

The following shows how SVRF specification statements are translated into TVF.

SVRF:

```
Layout System GDSII          // (1) use of case insensitivity
Layout Path $input_layout    // (2) use of environment variable for pathname
Layout Primary TOP

Source System SPICE           // (3) "source" is a Tcl command
Source Path $input_source
Source Primary TOP
```

TVF:

```
tvf::layout System GDSII
# (1) use all upper- or lowercase letters for command names;
# "System GDSII" are parameters passed to the tvf::layout command and are
# not case-sensitive
tvf::LAYOUT PATH $env(input_layout)
# (2) different environment variable mechanism
tvf::LAYOUT PRIMARY TOP
```

```
tvf::SOURCE System SPICE
# (3) SOURCE must be uppercase to avoid collision with Tcl source command
tvf::SOURCE PATH $env(input_source)
tvf::SOURCE PRIMARY TOP
```

## SVRF Statements Not Mapped to TVF

The PEX Inductance family of statements is not mapped to TVF. This is the only family of specification statements for which this is true.

In some cases in [Table 6-2](#), specification statements and layer operations share the same initial keyword. For example, FLATTEN is a layer operation, but FLATTEN INSIDE CELL is a specification statement. The specification statements are mapped to TVF, but the layer operations generally are not. In certain cases where layer operations are mapped to compile-time TVF (such as with COPY, DFM, and MDP), there generally is no need to use these layer operation commands because layer operation names are used inside of tvf::SETLAYER or tvf::RULECHECK constructs.

## Comment Characters

SVRF has three comment characters that are mapped to compile-time TVF commands as shown in [Table 6-3](#).

**Table 6-3. Comment Characters for Compile-time TVF**

Comment Type	SVRF Characters	TVF Comment Command
Checktext comment	@	tvf::@ or tvf::COMMENT
Single-line comment	//	tvf://
Block comment	/* ... */	no equivalent, but may be replicated with tvf::VERBATIM

You must include a whitespace character after the TVF command name.

## Layer Definitions and Assignments

In SVRF, layer definitions assign names to derived layers using this syntax:

*layer\_name* = *layer\_operation*

For example:

```
gate = poly AND diff
```

Layers defined in this way have user-specified names, so this syntax cannot be duplicated in TVF as a command. Instead, this is mapped in TVF by the SETLAYER command:

**tvf::SETLAYER layer\_name = arguments**

For example:

```
tvf::SETLAYER lout = ((L1 AND L2) OR "($L3 AND $L4)") AND {(L5 OR L6)}
```

Double quotes (" ") allow for variable substitution in a list of arguments, whereas braces ({} ) pass their contents as a list with no variable substitution.

The layer operations specified to SETLAYER use the same syntax as corresponding SVRF layer operations. Layer operations are passed as arguments to the SETLAYER command, and are then passed to the SVRF compiler. The return value from SETLAYER command is the name of the layer.

### Example 6-3. Layer Definitions and Assignments

The following are layer definitions and assignments that are global in scope in a compile-time TVF script:

```
tvf::LAYER POLY      1
tvf::LAYER NPOLY    31
tvf::LAYER PPOLY    32
tvf::SETLAYER ALLPLY = { ( POLY OR NPOLY ) OR PPOLY }
# equivalent to ALLPLY = ( POLY OR NPOLY ) OR PPOLY

set GR1 3.4
# usual tcl variable syntax

tvf::SETLAYER LARGEPLY = "SIZE ALLPLY BY $GR1"
# double quotes allow variable substitution
```

### Example 6-4. Layer Derivation

This example shows how SVRF syntax can be translated to TVF for layer derivation.

SVRF:

```
VARIABLE grid .23
CR0 = OPC OR NOOPC
CR18 = SIZE CR0 BY grid
//ADDED BIAS
CLTEST = EXTERNAL CR18 CR0 < .70 OPPOSITE EXTENDED 1.1 REGION CENTERLINE
CX1 = SIZE CR18 BY 2*grid
CC1 = CX1 NOT CLTEST
```

TVF:

```
set grid .23
tvf::SETLAYER CR0 = "OPC OR NOOPC"
tvf::SETLAYER CR18 = "SIZE CR0 BY $grid"
tvf:// ADDED BIAS
tvf::SETLAYER CLTEST = "EXTERNAL CR0 CR18 < .70 OPPOSITE EXTENDED 1.1 \
REGION CENTERLINE"
```

```
#use of line continuation character
tvf::SETLAYER CX1 = "SIZE CR18 BY [expr 2 * $grid]"
tvf::SETLAYER CC1 = "CX1 NOT CLTEST"
```

## SETLAYER Options in Compile-Time TVF

SETLAYER keeps track of layers defined globally as well as locally within a tvf::RULECHECK command (the RULECHECK command is discussed under [Rule Checks](#)), and can be queried for layer names through the -getnames, -local, -global, and -getexpr switches. For example,

```
set layerList [tvf::SETLAYER -getnames -global]
```

assigns to variable *layerList* a list of all the layer names that appear as the first argument of the SETLAYER commands occurring outside of RULECHECK commands. The corresponding syntax for retrieving the layer names of SETLAYER commands within a RULECHECK command is this:

```
set layerList [tvf::SETLAYER -getnames -local]
```

To get the layer operation expression for a derived layer previously set by an SETLAYER command, use the -getexpr option:

```
set layer_exp [tvf::SETLAYER layer_name -getexpr]
```

For example, if *layer\_name* were ALLPLY, which is defined this way:

```
tvf::SETLAYER ALLPLY = {POLY2 OR NPOLY2}
```

then

```
set ALLPLY_def [tvf::SETLAYER ALLPLY -getexpr]
```

assigns “POLY2 OR NPOLY2” to ALLPLY\_def.

## Rule Checks

Rule checks in SVRF have user-given names, so the SVRF syntax cannot be used in TVF. Instead, this construct is handled by a tvf::RULECHECK command:

```
tvf::RULECHECK name {
    tvf::SETLAYER name = arguments
    tvf::OUTLAYER arguments
    tvf::COPY layer
}
```

The three commands shown can be used inside the body of tvf::RULECHECK.

The SETLAYER command was defined in the section “[“Layer Definitions and Assignments” on page 1896](#)”. In this context it creates a layer that is local in scope to the RULECHECK command in which it appears.

The tvf::OUTLAYER command sends the results of its arguments to the nmDRC results database. For example, the following SVRF rule check:

```
my_rule {
    layer1 = layer2 OR layer3
    layer4 NOT layer1
}
```

would be coded like this in TVF:

```
tvf::RULECHECK my_rule {
    tvf::SETLAYER layer1 = {layer2 OR layer3}
    tvf::OUTLAYER {layer4 NOT layer1}
}
```

The SETLAYER command arguments may be enclosed in double quotes or braces. Both of these grouping elements may appear in the same group of arguments, similar to the SETLAYER command. Again, double quotes allow variable substitution and braces do not.

The third layer command allowed in a RULECHECK statement is the tvf::COPY command. This is the only layer operation mapped in TVF, and it performs the same function as its SVRF counterpart. COPY does not need to be embedded in an OUTLAYER command.

Remember, the layer operation syntax is identical to SVRF, and is passed as arguments to the RULECHECK command.

Rule check comments can be specified either through the COMMENT command, or through the @ command. Note that in the latter form, the @ sign also has to be followed by whitespace. Also see the command tvf::set\_rule\_check\_indentation in [Table 6-4](#) for customizing your SVRF output when saving it to a file.

### Example 6-5. for Statement

This is an example of a for loop, showing basic checks for four layers of vias.

```
for {set i 1} {$i <= 4} {incr i} {
    tvf::RULECHECK VIA${i}_SIZE {
        tvf::@ Exact via$i size is 0.36 by 0.36
        tvf::OUTLAYER "NOT RECTANGLE via$i == 0.36 BY == 0.36"
    }

    tvf::RULECHECK VIA${i}_SPACE {
        tvf::@ Exact via$i spacing is 0.35
        tvf::OUTLAYER "EXT via$i < 0.35 ABUT < 90 SINGULAR"
    }

    tvf::RULECHECK METAL${i}_ENCLOSE_VIA$i {
        tvf::@ Minimum metal$i enclosure of via$i is 0.02
        tvf::OUTLAYER "ENC via$i metal$i < 0.02 ABUT OUTSIDE ALSO"
    }
}
```

### Example 6-6. foreach Statement

This example shows a foreach loop, which passes width, spacing, and enclosure parameters for five layers of metal.

```
foreach {i} [list 1 2 3 4 5] \
    {width} [list 0.03 0.4 0.4 0.4 0.44] \
    {space} [list 0.25 0.4 0.4 0.4 0.46] \
    {enclose} [list 0.01 0.02 0.02 0.02 0.09] {

    tvf:::// ***** METAL ${i} CHECKS ***** //

    tvf::RULECHECK METAL${i}_WIDTH {
        tvf::@ Minimum Metal ${i} width = $width
        tvf::OUTLAYER "INTERNAL met${i} < $width ABUT < 90 SINGULAR"
    }

    tvf::RULECHECK METAL${i}_SPACE {
        tvf::@ Minimum Metal ${i} space = $space
        tvf::OUTLAYER "EXTERNAL met${i} < $space ABUT < 90 SINGULAR"
    }

    set im1 [expr $i - 1]
    if { $im1 > 0 } {
        tvf::RULECHECK METAL${i}_ENCLOSE_VIA$im1 {
            tvf::@ Minimum Metal ${i} enclose Via $im1 = $enclose
            tvf::OUTLAYER "ENCLOSURE via${im1} met${i} < $enclose ABUT \
                OUTSIDE ALSO"
        }
    }
}
```

### Example 6-7. Tcl proc

This example shows the definition of a Tcl proc. The procedure in this example shows width and spacing checks. The calls to the procedure at the end of the example pass the values for the layer, Space, and Width variables.

```
#verbatim SVRF code to assign and define layers
tvf::VERBATIM {
    bulk = EXTENT
    LAYER pwell 9
    LAYER diff 10
    LAYER nplus 11
    LAYER pplus 12

    nwell = bulk NOT pwell
    ndiff = nplus AND diff
    pdiff = pplus AND diff
}

#macro function to do basic width and spacing checks
proc Space_Width_Check {layer Space Width} {
    set LAYER [string toupper $layer];
    #convert layer to uppercase
```

```

tvf:::// ***** $LAYER CHECKS ***** //

tvf:::RULECHECK ${LAYER}_WIDTH {
    tvf:::@ Min $layer width is $Width
    tvf:::OUTLAYER "INT $layer < $Width ABUT < 90 SINGULAR"
}

tvf:::RULECHECK ${LAYER}_SPACE {
    tvf:::@ Min $layer space is $Space
    tvf:::OUTLAYER "EXT $layer < $Space ABUT < 90 SINGULAR"
}
}

#check nwell and diffusion

Space_Width_Check nwell 0.80 0.80
Space_Width_Check pdiff 0.40 0.30
Space_Width_Check ndiff 0.40 0.30

```

## Handling of Special Characters

It is generally easiest to pass the runtime TVF elements in a compile-time script using the `tvf::VERBATIM` command. However, the runtime element TVF Function is implemented as a compile-time command and can appear outside of a `VERBATIM` command.

Because of conflicting interpretations of special characters between Tcl and SVRF, care is needed in the handling of grouping characters, especially when runtime TVF statements appear in a compile-time script. The [ ], and \$ characters, in particular, need special handling. This is shown in the following example:

```

#! tvf
...
tvf:::TVF FUNCTION ERROR errorTest { [ /*
... user runtime Tcl script
... } "$evaluate_tvf_compile_time_variable" { ...
... more user runtime Tcl script
*/ ]
}

```

The usual syntax for the runtime TVF Function element is discussed on [page 1906](#). In this example, the braces { } are used to protect the square brackets [ ] from premature evaluation in the compile-time environment. The C-style comment characters /\* ... \*/ are used in the runtime phase to protect embedded square brackets from the rule file compiler. This is discussed under [“Coding Guidelines”](#) on page 1909.

Continuing this example, this shows further use of double quotes and backslash characters:

```

tvf:::RULECHECK test_result {
    TVF errorTest POLY DIFF "\[ /* 
        <passed parameters>
*/ \]"

```

This shows the typical `tvf:::RULECHECK` syntax with the runtime TVF layer operation (see [page 1908](#) for details on this operation) embedded. The double quotes are used to pass a list of

parameters to the TVF layer operation. The backslashes are used to protect the square brackets from premature evaluation in the compile-time environment. The C-style comment characters are used for reasons described previously in this section.

Again, it is easier to pass runtime syntax using the VERBATIM command so you do not have to be concerned about the special characters.

## Inserting Quotes in Pathnames

In some cases, it may be desirable to add quotation marks for pathnames in commands. For example:

The following TVF command:

```
tvf::INCLUDE filename.tvf
```

results in this SVRF statement:

```
INCLUDE filename.tvf
```

In order to cause the resulting filename to be enclosed in double quotes, add the escaped quotes in the TVF code:

```
tvf::INCLUDE "\\"filename.tvf\\\""
```

This is the SVRF result:

```
INCLUDE "filename.tvf"
```

## Passing Command Line Arguments in Compile-Time TVF

Command-line arguments can be passed into a compile-time TVF script by using the -tvfarg command-line option:

```
calibre -tvfarg my_argument
```

Note that the argument (*my\_argument*) cannot contain spaces. In the compile-time TVF script, *my\_argument* can be accessed as follows:

```
set tvf_arg [tvf::get_tvf_arg]
```

If no -tvfarg option is given on the command line, tvf::get\_tvf\_arg returns an empty string. The following are examples of valid command-line usage:

```
calibre -tvfarg my_argument -drc -hier rules
calibre -drc -hier -tvfarg my_argument rules
```

---

**Note**

 The tvf::get\_tvf\_arg command replaces tvf::vars(tvfarg) starting with the 2006.3 release.

---

## Compile-Time TVF File Usage

To implement the programmable mode, Calibre detects the presence of a compile-time TVF script by examining the first line of the specified rule file. If the first line consists of the string “#!tvf”, the entire rule file is read by the TVF preprocessor.

There are two methods for loading a compile-time TVF rule file into Calibre. The first method interprets the TVF rule file and generates an internal SVRF output. The SVRF output is then fed to the SVRF compiler, one line at a time. Upon successful compilation of the SVRF output, the Calibre job is run. The shell command for this method is:

```
calibre <usual calibre options> tvf_rule_file
```

which is exactly the same as any typical Calibre command line execution. However, the following line is echoed to stdout, indicating a TVF rule file is in use:

```
Rule file is in Tcl Verification Format (TVF), converting to SVRF.
```

The TVF rule file is written to the transcript, followed by the runtime information. The translated SVRF code is not written to the transcript by default. To enable the echoing of the SVRF code to stdout, add the following line to the TVF file:

```
tvf::echo_svrf 1; # echo generated SVRF statements
```

---

### Note

 The tvf::echo\_svrf command replaces the use of “set tvf::vars(output) stdout” starting with the 2006.3 release.

---

You can optionally specify an additional argument to write to a file instead of to stdout. For example:

```
set externalChannel [open "out.svrf" w]
tvf::echo_svrf 1 $externalChannel

tvf::VERBATIM {
...
}

tvf::echo_svrf 0
```

The generated SVRF statements are printed first, and then the TVF input file is printed. If there is an error in the TVF preprocessing phase, the TVF input file is not printed.

The environment variable CALIBRE\_ECHO\_RULE\_FILE, when set to any value, causes the generated SVRF statements to be printed after any SVRF Include statements have been processed. In this echo mode, the TVF input file is not printed.

The second method for loading a TVF rule file runs the Tcl preprocessor on the TVF rule file and outputs the internally-generated SVRF code to the output file specified after the -E option. The output file is not loaded into the compiler, nor is a Calibre job initiated. The command for the preprocessor-only mode is:

```
calibre -E svrf_output_file tvf_input_file
```

The commands in [Table 6-4](#) are designed to aid you in customizing SVRF output generated using preprocessor-only mode. Each of these commands can be embedded in a compile-time TVF script and are applied to the output file when your script is translated to SVRF using the Calibre -E command-line option.

**Table 6-4. Preprocessor-Only Commands**

Command Name	Description
<code>tvf::echo_to_svrf_file <i>string</i></code>	Writes a <i>string</i> directly to the output SVRF file.
<code>tvf::set_rule_check_indentation <i>indentation</i></code>	Controls indentation of each line printed in the output SVRF file (the default is a tab). The <i>indentation</i> parameter is a string that is pre-pended to each line. If <i>indentation</i> is an empty string (""), lines are not indented.
<code>tvf::set_disclaimer <i>disclaimer</i></code>	Writes a formatted header, specified with the <i>disclaimer</i> string, to the top of the SVRF output file and replaces the default header. This command, if used, must be the first command specified in your TVF file.

## Identifying Errors in Compile-Time TVF

Calibre runs compile-time TVF with targeted error reporting with the `-tvf_instrument` command-line option. When it is used, TVF errors are reported with the file name and line number. If an error occurs in SVRF code generated out of compile-time TVF, the original TVF location that produced the offending SVRF statement is reported.

An example of a TVF source file error produced with targeted error reporting enabled is shown in the transcript below. The “ERROR” notation on the initial line of the transcript indicates where the error occurred and cites the command with the syntax violation.

```
ERROR: Error TVF1 on line 28 of rules.tvf - TVF preprocessor error: ERROR
-- invalid command name "::NONEXISTENT_OUTPUT"
      while executing
"NONEXISTENT_OUTPUT { VIA AND CONT} "
      invoked from within
":::tvf::RULECHECK VIACNT27 {NONEXISTENT_OUTPUT { VIA AND CONT} }"
      invoked from within
"RULECHECK VIACNT27 {NONEXISTENT_OUTPUT { VIA AND CONT} }"
      invoked from within
"if { 1 } {
      RULECHECK VIAWDT27 {OUTPUT { INT VIA < 2 REGION} }
      RULECHECK VIATVI27 {OUTPUT { EXT VIA < 2 REGI...
(file "rules.tvf", line 28)
```

The drawback to using the `-tvf_instrument` command-line option is that the initial scan detects all errors, even in portions of the file that would normally not be executed. If the instrumenter were to be permanently engaged, this would make Calibre incompatible with previous versions.

Errors found by the syntax checker are legitimate Tcl syntax errors. However, the end user may not be capable of correcting the erroneous code, especially if the error exists in TVF libraries provided by a third party.

If compile-time TVF code is run without the `-tvf_instrument` command line option, the `tvf::set_traceback_line` command can still be used to identify the SVRF line with syntax errors, but the preferred method of finding syntax errors is with the `-tvf_instrument` command-line option.

## Identifying Errors Without Targeted Error Reporting

SVRF syntax errors can be traced without the `-tvf_instrument` command-line option. This can be done by tracing the line number reported by the SVRF compiler and using the `tvf::set_traceback_line` command. For example, assume this error is reported by the SVRF compiler:

```
ERROR: Error INP3 on line 52 of SVRF generated from TVF
```

You add this line to your TVF code:

```
tvf::set_traceback_line 52; # force trace at line 52
```

This instructs the Tcl preprocessor to issue an error message when the output line number is equal to 52. When you run Calibre again on the TVF file that contains this command, the TVF line number is reported as shown here:

```
ERROR: Error TVF1 on line 79 of rules.tvf - TVF preprocessor error:  
Traceback forced at SVRF output line number 52
```

This tells you the error is on line 79 of the TVF file. The `tvf::set_traceback_line` command replaces the use of “`set tvf::traceback_line_number`” starting with the 2006.3 release.

---

### Note

 Prior to 2009.1, the `tvf::set_traceback_line` function was used to identify the SVRF line with syntax errors. Starting in 2009.1 and later, the `-tvf_instrument` command-line option should be used to find syntax errors in compile-time TVF.

---

## Runtime TVF

Runtime TVF is Tcl code that appears within a special SVRF element called TVF Function.

There are two main uses of the TVF Function call:

- To define a function called by the SVRF layer operation called **TVF**. The TVF layer operation with runtime TVF allows control of layer output based upon manipulating data that is generated during a Calibre run.
- To define a Tcl procedure or set of procedures called by other Calibre processes. The Device property computation language supports calls to Tcl procedures defined in TVF Function statements (see [Summary of Device Property Computation Functions](#)). Calibre PERC (Programmable Electrical Rule Checking) uses the TVF Function statement to define all of its rule check procedures; see the [Calibre PERC User's Manual](#).

The TVF Function and TVF layer operation statements can appear in a normal SVRF rule file because they are SVRF elements themselves.

These elements may be placed in a compile-time TVF script (#!tvf). The easiest way of doing this is within a tvf::VERBATIM command (see [“Passing Verbatim SVRF Code in Compile-Time TVF”](#) on page 1893).

Note that runtime TVF does not support any operations that require connectivity extraction to occur.

## TVF Function

The SVRF element TVF Function defines a function block of Tcl code. When used with the TVF layer operation, the function block is Tcl code that returns layer data.

When not used with the TVF layer operation, the TVF function block defines a set of Tcl procedures that optionally return a numeric value or a string.

Each TVF Function is independent of any other TVF Function in the same rule file; hence, there is no communication between separate TVF function blocks.

### Syntax

```
TVF FUNCTION [EDGE | ERROR] tvf_name [  
    inline TVF runtime script or Tcl procs  
]
```

### Parameters

- **TVF FUNCTION**

Required keyword that begins a runtime TVF procedure.

- EDGE | ERROR

Optional keyword that specifies the type of layer to output when defining TVF layer operations. By default, the output is a derived polygon layer, and no additional keyword is needed. For other output types, you can specify the following:

EDGE — Specifies the output of the function is a derived edge layer.

ERROR — Specifies the output of the function is a derived error layer. Only the Enclosure, External, and Internal layer operations produce this type of output for runtime TVF.

When defining TVF layer operations, the type of layer output actually generated by the TVF Function must match what you specify for the output type. This is checked at runtime.

- *tvf\_name*

Required name of the function. Each TVF Function must have a unique *tvf\_name*. The *tvf\_name* is a string that follows the same conventions as the name of a SVRF rule check.

- [

*inline TVF runtime script or Tcl procs*

]

Required function block containing runtime script or Tcl procs appearing between literal square brackets. The function block must contain valid Tcl code. The brackets must appear on separate lines from the function block.

The TVF commands that can appear in this function block for DRC-style checks are described under “[Runtime TVF Commands](#)” on page 1911.

TVF commands used for device recognition are described under “[TVF Functional Interface for Device Property Calculation](#)” on page 1827.

## Example

Here is an example of using TVF Function, but without a specific function block:

```
TVF FUNCTION my_test_func /*  
    Tcl_code  
 */ ]
```

The example shows the use of C-style comment characters with the opening and closing brackets of TVF Function. This is necessary to avoid collisions between the SVRF and Tcl interpretation of brackets within the Tcl code block. This is explained in more detail in [Coding Guidelines](#).

For a complete example of using TVF Function with a TVF layer operation, see [Example 6-8, Runtime TVF](#).

For a complete example of using TVF Function with the Device property language, see [TVF Functional Interface for Device Property Calculation](#) and “[Example Device Property Computation with TVF Function](#)” on page 1828.

See “[PERC Example Rule File](#)” in the *Calibre PERC User’s Manual* for a Calibre PERC example using TVF Function.

## Performance Notes and Multithreading

In multi-threaded runtime TVF execution, only one Tcl interpreter is created and all access to the interpreter is sequential (single-threaded). This means TVF FUNCTION blocks are processed one-at-a-time. This does not apply to SVRF portions of a rule file that contain elements that support multi-threading. Be sure to use standard Tcl performance coding practices such as the use of {} in expr statements. Also be aware that operations like list lookup, list sorting, and management of complex data structures may take significant time.

## TVF Layer Operation

The TVF layer operation behaves similarly to any SVRF layer operation in that it is used within a rule check or in a layer derivation. The TVF layer operation calls a TVF Function and passes layer parameters to it, then generates a single output layer based upon one or more input layers. The main difference between the TVF layer operation and other layer operations is the TVF layer operation serves as a call to a TVF Function. A TVF layer operation may not appear within a TVF Function. The TVF layer operation is single-threaded only.

Here is the TVF layer operation syntax:

**TVF** *tvf\_name* *layer1* [ *layerN* ... ] [ ‘[  
    *Tcl initialization code*  
]’]

The parameters are as follows:

- **TVF**  
Required keyword indicating a TVF layer operation.
- ***tvf\_name***  
Required parameter that refers to a TVF Function of the same name.
- ***layer1* [ *layerN* ... ]**  
One or more required input layers. These layers are passed as arguments to the TVF Function *tvf\_name*, and they must be of the appropriate input layer type (original, derived polygon, or derived edge) that are used by the operations appearing in the TVF Function *tvf\_name*.

---

### Caution



No flattened layers may be used as a layer input to TVF.

---

- [  
*Tcl\_initialization\_code*  
 ]

An optional Tcl script that is executed before the code within the function block of *tvf\_name*. You can set global variables, for example, in this code section. If you specify a *Tcl\_initialization\_code* section, it must appear between square brackets, and the brackets must appear on lines by themselves.

### Example 6-8. Runtime TVF

This is a simple example to introduce the forms of the SVRF elements supporting runtime TVF.

```
TVF FUNCTION example_function /*  

    # This function generates polygon-directed output.  

    # The C-style comment character is used to prevent the SVRF  

    # compiler from complaining about embedded square brackets [ ].  

    tvf:::GET_LAYER_ARGS layer1 layer2  

    # create the input layer handles and get the layers passed from the  

    # TVF layer operation calling this function  

    tvf:::SETLAYER one_not_two = "$layer1 NOT $layer2"  

    # perform a NOT operation on the input layers  

    # check to see if the derived layer exists, then perform output  

    if {[tvf:::IS_LAYER_EMPTY $one_not_two]} {  

        tvf:::OUTLAYER "COPY $layer1"; # one_not_two is empty  

        puts "$layer1 is coincident with $layer2: outputting $layer1"  

    } else {  

        tvf:::OUTLAYER "COPY $one_not_two"; #one_not_two is not empty  

    }  

*/] // end of function  

// The C-style comment character is used to prevent the SVRF  

// compiler from complaining about embedded square brackets [ ].  

my_rule { TVF example_function vial via2 }  

// use vial and via2 as input layers to example_function and return the  

// output to the nmDRC results database
```

You could also send the output to a derived layer, like this:

```
out = TVF example_function vial via2
```

## Coding Guidelines

Embedded square brackets [ ] in Tcl commands within a TVF Function block cause the SVRF compiler to report a syntax error (SYN2) at the *initial line number* of the TVF Function. A similar error occurs for embedded brackets in a TVF layer operation. This is because the square brackets are special characters in SVRF. This error can be avoided in several ways.

The most elegant solution is to use C-style /\* ... \*/ comment characters on the same lines as the square brackets required by the TVF Function syntax, as shown here:

```
TVF FUNCTION example_function1 /*  
    <inline Tcl script containing square brackets>  
*/ ]
```

Here is an example for the TVF layer operation:

```
TVF example_function1 my_layer1/*  
    [Tcl initialization script containing square brackets]  
*/ ]
```

This causes the SVRF compiler to pass any embedded square brackets in the Tcl commands without causing an error (square brackets in SVRF layer operations must be escaped with a backslash character). If you want to use this protection method for separate sections of your script, you can do the following:

```
TVF FUNCTION example_function /*  
    [<inline TVF script>]  
    #*/ end of the C-style comment block;  
    <unprotected TVF code>  
    #/* begin another protected block;  
    [<inline TVF script>]  
*/ ]
```

Note that nested C-style comments ( /\* /\* \*/ \*/) cause an SVRF compiler error.

Another method used to protect individual lines of code that contain square brackets is shown in this example:

```
TVF FUNCTION example_function [  
    if {0} {'}; [<code containing square brackets>]  
    # The single quote causes the SVRF compiler to pass square brackets.  
    if {0} { // }; [<code containing square brackets>]  
    # Same effect.  
]
```

The method shown in this previous example has the additional advantage of protecting nested C-style /\* ... \*/ comments.

## Layers in Runtime TVF

Layers in runtime TVF are variable handles. [Runtime TVF Example 6-8](#) shows this, where the \$layer1 and \$layer2 variables appear. Layer handles are used by various commands within the TVF Function that manipulate layer data. Copies of a layer handle all point to the same layer. Any temporary layers that you store in a variable, or any implicitly derived layers, are deleted automatically when they are no longer needed.

Calibre checks the layer arguments for the appropriate type at runtime. The layers you pass to a TVF Function must be of the correct type for the layer operations that receive them as input. Similarly, the layer type that is output from a TVF Function must match the type specified by the output keywords: EDGE, ERROR, or no output keyword (the default), which corresponds to polygon layer data.

## Runtime TVF Commands

General runtime TVF command syntax is identical to compile-time syntax, as shown under [“Command Syntax”](#) on page 1892. TVF commands are case-sensitive, and both upper- and lowercase command names are available.

## Layer Functions

[Table 6-5](#) shows the layer functions that can appear in a runtime TVF inline script.

**Table 6-5. Runtime TVF Layer Functions**

Function	Description
<code>tvf::DELETE_LAYER \$layer</code>	Deletes the layer specified by <code>\$layer</code> . Subsequent references to <code>\$layer</code> return an error. Most useful for reducing peak memory usage by removing layers from the run.

**Table 6-5. Runtime TVF Layer Functions (cont.)**

Function	Description
<code>tvf::GET_LAYER_ARGS <i>layer1</i> [<i>layerN</i> ...] [<i>args</i>]</code>	<p>Used to import the layers passed to a TVF Function by a TVF layer operation. Sets up the layer handles \$layer1, \$layer2, and so forth. The number of imported layer arguments must match the number of layer arguments passed by the TVF layer operation.</p> <p>The optional args parameter is a literal name used to reference a list of layer arguments passed from a TVF layer operation. If args is specified, it must be the final argument in this command.</p> <p>Must be used once in the TVF Function called by a TVF layer operation.</p>
<code>tvf::GET_LAYER_EXTENT \$<i>layer</i></code>	Returns the bounding box of the extent of the <i>layer</i> as a list {x1 y1 x2 y2}.
<code>tvf::IS_LAYER_EMPTY \$<i>layer</i></code>	Used to test for an empty layer. Returns true (1) if \$layer is empty.
<code>tvf::OUTLAYER <i>arguments</i></code>	<p>Executes the <i>arguments</i> (which are layer operations) and returns the layer to the TVF layer operation that called the TVF Function.</p> <p>Must be used once in the TVF Function called by a TVF layer operation, and the output layer type must match the output type specified for the TVF Function.</p>
<code>tvf::SETLAYER <i>layer_name</i> = <i>arguments</i></code>	Executes the <i>arguments</i> (which are layer operations) and stores the output in the <i>layer_name</i> handle. Used for layer derivation. (This command is discussed in <a href="#">Layer Definitions and Assignments</a> . The behavior is similar to the compile-time environment.)

A TVF Function that is called by a TVF layer operation statement must have exactly one `tvf::GET_LAYER_ARGS` function and one `tvf::OUTLAYER` function.

The “args” argument for tvf::GET\_LAYER\_ARGS can be used like this:

```
x = TVF my_function poly1 metal1 metal2 metal3

TVF FUNCTION my_function [
    tvf::GET_LAYER_ARGS P1 args
    # $args contains poly1 metal1, metal2 and metal3
    ...
]
```

The other functions in [Table 6-5](#) can be used as needed. The tvf::SETLAYER command is frequently used to derive intermediate layers. A basic runtime TVF example is shown in [Example 6-9](#), and two more complicated examples are given in [Example 6-10](#) and [Example 6-11](#).

### **Example 6-9. Simple TVF Function and TVF Layer Operation Call**

Here, the L1, L2, and L3 arguments correspond to layer1, layer2, and layer3.

```
TVF FUNCTION my_function [
    tvf::GET_LAYER_ARGS L1 L2 L3
    tvf::SETLAYER l_int = "$L1 AND $L2"
    tvf::OUTLAYER "NOT INSIDE l_int $L3"
]

x = TVF my_function layer1 layer2 layer3
```

## **Layer Operations**

[Table 6-6](#) shows the SVRF layer operations you can use in runtime TVF code:

**Table 6-6. Layer Operations in Runtime TVF**

AND <sup>1</sup>	[Not] Enclose Rectangle	Merge	Snap
[Not] Angle	Enclosure <sup>2</sup>	NOT	[Not] Touch
[Not] Area	Expand Edge	OR <sup>1</sup>	[Not] Touch Edge
[Not] Coincident Edge	Extent	[Not] Outside	[Not] Touch Inside Edge
[Not] Coincident Inside Edge	Extents	[Not] Outside Edge	[Not] Touch Outside Edge
[Not] Coincident Outside Edge	External <sup>2</sup>	Path Length	Vertex
Convex Edge	Grow	Perimeter	[Not] With Edge
Copy	Holes	Push	[Not] With Neighbor
[Not] Cut	[Not] Inside	[Not] Rectangle	[Not] With Width

**Table 6-6. Layer Operations in Runtime TVF (cont.)**

Density	[Not] Inside Edge	Rectangles	XOR <sup>1</sup>
DFM RDB	[Not] Interact <sup>2</sup>	Rotate	
[Not] Donut	Internal <sup>2</sup>	Shrink	
[Not] Enclose	[Not] Length	Size	

1. Single-layer Boolean operations are not supported.
2. Keywords requiring connectivity extraction are not supported.

These operations are executed within tvf::SETLAYER or tvf::OUTLAYER commands, and are passed as parameters to these commands. They are case-insensitive and the standard abbreviations, for the operations that have abbreviated forms, may be used.

SVRF allows you great freedom in layer operation parameter order, while TVF is somewhat more restrictive. Here are a few restrictions to be aware of in runtime TVF:

- The primary keyword set of a layer operation must precede any secondary keywords.

For example, in runtime TVF you may write this:

```
EXT met1 < 0.1 ABUT < 90
```

or this:

```
EXT ABUT < 90 met1 < 0.1
```

but not this:

```
ABUT < 90 EXT met1 < 0.1
```

because EXT is a primary keyword and ABUT is a secondary keyword.

- If a layer operation has a constraint parameter, the constraint must immediately follow the layer parameters. The previous example shows the correct order for layer and constraint parameters.
- As previously discussed, the square bracket ([ ]) operators used in the Enclosure, External, and Internal operations must be escaped with a leading backslash (\). For example:

```
EXT \[ met1\] < 0.1
```

## Using Variables

Variables can be set in a rule file using the SVRF [Variable](#) specification statement. Rule file variables are accessed in runtime TVF using the tvf::svrf\_var function, which has this syntax:

**tvf::svrf\_var *variable\_name***

Any variable name that is declared in a Variable statement in the rule file is accessible in a TVF Function using this construct.

Here is a simple example:

```
VARIABLE myVar1 0.1
...
TVF FUNCTION myTVFscript [ /*
    tvf::GET_LAYER_ARGS L1 L2
    set dist [tvf::svrf_var myVar1]
    tvf::SETLAYER ext1 = "EXTERNAL $L1 $L2 <= $dist REGION OPPOSITE"
    ...
*/ ]
layer1 = TVF myTVFscript A B
```

You can accomplish the same thing with the added flexibility of reassigning variable values using global Tcl variables and the initialization section of the TVF layer operation, as shown here:

```
TVF FUNCTION myTVFscript [ /*
    tvf::GET_LAYER_ARGS L1 L2
    global globalVar1
    set dist $globalVar1
    tvf::SETLAYER ext1 = "EXTERNAL $L1 $L2 <= $dist REGION OPPOSITE"
    ...
*/ ]
layer1 = TVF myTVFscript A B [
    global globalVar1
    set globalVar1 0.1
]
layer2 = TVF myTVFscript C D [
    global globalVar1
    set globalVar1 0.2
]
```

## System Variables

There are a number of system variables declared in the rule file that runtime TVF can access. These are listed in Table 6-7:

**Table 6-7. Accessible System Variables**

Variable Name	Description
The following variables can be used outside of a TVF Function in an SVRF rule file.	
PRECISION	Resolves to the rule file precision value (which may or may not have been explicitly defined by a <a href="#">Precision</a> specification statement).
PRECISION_N	Resolves to the numerator of the rule file Precision value, if expressed as a ratio; otherwise, resolves to the rule file Precision.

**Table 6-7. Accessible System Variables (cont.)**

Variable Name	Description
PRECISION_D	Resolves to the denominator of the rule file Precision value, if expressed as a ratio; otherwise, resolves to 1.
The following variables may only be used in a TVF Function.	
RESOLUTION	Resolves to the value defined with the <a href="#">Resolution</a> specification statement when a single argument is specified, or two equal arguments are specified.
RESOLUTION_X	Resolves to the first value defined with the Resolution specification statement when two arguments are specified.
RESOLUTION_Y	Resolves to the second value defined with the Resolution specification statement when two arguments are specified.
LAYOUT_PATH	Resolves to the pathnames specified with the <a href="#">Layout Path</a> specification statement.
LAYOUT_PRIMARY	Resolves to the top-level cell, design filename of the layout, or subcircuit specified with the <a href="#">Layout Primary</a> specification statement.

System variables are accessed with the following command:

***tvf::sys\_var system\_variable***

If the system variable does not exist, tvf::sys\_var returns an empty string ("").

The values of system variables can be accessed as shown in this example:

```
// accesses the value of PRECISION
set precision [tvf::sys_var PRECISION]
```

---

**Note**

 Use of tvf::sys\_var to access system variables replaces the use of tvf::vars( ) starting with the 2006.3 release.

---

## Comment Character

The only comment character allowed within a runtime TVF script is #.

The use of C-style /\* ... \*/ comment characters within your TVF Function script to protect square brackets from the SVRF compiler is discussed under [“Coding Guidelines”](#) on page 1909.

## Examples

### Example 6-10. SVRF Versus TVF Layer Derivations

The following shows an example of testing some typical SVRF layer derivations against TVF derivations that should produce identical results.

```
LAYER POLY 3

// Derive layers to check
L1 = GROW POLY TOP BY 1
L2 = SHRINK L1 BOTTOM BY 1
L3 = GROW L2 LEFT BY 4.0
OVLP = SHRINK L3 RIGHT BY 3.0
EXT1 = EXTERNAL [POLY] OVLP <= 4

EXT1_CHK { TVF tvf_operator POLY EXT1 }
    // calls the tvf_operator function and passes POLY and EXT1 as layers

TVF FUNCTION EDGE tvf_operator /**
    # edge-directed output is expected
    # C-style comment characters protect square brackets in the script.

    tvf::GET_LAYER_ARGS poly_in ext1_in
        # create the input layer handles

    tvf::SETLAYER ovlp_new = "SHRINK (GROW (SHRINK \
        (GROW $poly_in TOP BY 1) BOTTOM BY 1) LEFT BY 4.0) RIGHT BY 3"
        # line continuation \ character is required
        # derives a layer using the same chain of operations as OVLP

    tvf::SETLAYER ext1_new = "EXTERNAL \\[$poly_in\\] $ovlp_new <= 4"
        # derives a layer in the same way as EXT1
        # Backslashes are needed to protect the literal brackets.

    tvf::SETLAYER delta = "(EXPAND EDGE $ext1_new BY .001) \
        XOR (EXPAND EDGE $ext1_in BY .001)"
        # derive delta to show any difference between EXT1 and ext1_new

    if {[tvf::IS_LAYER_EMPTY $delta]} {
        # square brackets needed for evaluation of a Boolean output function

        tvf::OUTLAYER "COPY $ext1_in"; # ext1_new is the same as EXT1
    } {
        puts "Error, results are not the same, returning SVRF results"
        tvf::OUTLAYER "COPY $ext1"
    }
*/] // end of function
```

### Example 6-11. while Statement

This example shows a while loop.

```
TVF FUNCTION tvf_loop [ /*

# The loop expands the DIFF layer until it covers the POLY level.
# It then writes out the final expanded DIFF layer.

# echo the calls to layer operations in the transcript
tvf::echo_svrf 1

tvf::GET_LAYER_ARGS diff poly; # get the input layer handles
set size_value $initial_size_value
    # assign passed-in parameter from TVF operation to new variable

while {1} {
    set size_value [expr $size_value + 0.5]

    # expand the diff layer and then check to see if it covers poly
    tvf::SETLAYER expanded_layer = "SIZE $diff BY $size_value \
        TRUNCATE 0.2"
    tvf::SETLAYER overlap = "$poly NOT $expanded_layer"

    # show if diff covers poly in the transcript; 0 means no
    puts "IS_LAYER_EMPTY \$overlap returns: [tvf::IS_LAYER_EMPTY $overlap]"

    # test to see if diff covers poly, if yes print the expansion
    # value and output the expanded layer.
    if {[tvf::IS_LAYER_EMPTY $overlap]} {
        puts "diff covers poly when expanded by $size_value"
        tvf::OUTLAYER "COPY $expanded_layer"
        break; # leave while loop
    }

    # delete these layers
    tvf::DELETE_LAYER $overlap
    tvf::DELETE_LAYER $expanded_layer

}; # end of while loop
*/] // end TVF FUNCTION

expand_diff { TVF tvf_loop DIFF POLY [
    set initial_size_value 0
]
}
```

### Example 6-12. Using Compile-Time TVF with Runtime TVF Code

Compile-time TVF can be used to pass runtime TVF syntax as is shown in this example. You pass the filename of a text file as a parameter to the TVF layer operation. This text file is then read by the TVF Function to establish variable settings within the function.

```
#!tvf
...
tvf::VERBATIM {
...
}
```

```
// using TVF layer operation to pass parameters, including a filename

tvf::SETLAYER out_layer = TVF my_function inlayer1 inlayer2 {[[
    set filename "parameters.txt"
]}

TVF FUNCTION my_function /* 
tvf::GET_LAYER_ARGS layer1 layer2; # get the input layer handles

# read the contents of the parameters file

set filein $filename
set infile [open $filein]
while { ! [eof $infile]} {
    set tmp [gets $infile]
    lappend listin $tmp
}
...
<commands using $listin values>
...
*/ ]
}
```

## Echoing Layer Operation Calls to stdout

Calibre normally prints the SVRF lines that are specified by the rule file after they are loaded into the SVRF compiler. However, runtime TVF code is not processed by the SVRF compiler; therefore, calls to layer operations made from a TVF Function are not output by default. To enable the echoing of all layer operation calls to stdout, add the following line to the TVF Function script:

```
tvf::echo_svrf 1; # echo layer operation calls to stdout
```

This shows all calls to layer operations in sequentially-numbered lines like this:

```
--- TVF:1 tvf::SETLAYER "SIZE @0 BY 0.5 TRUNCATE 0.2"
--- TVF:2 Calling Drc_get_temporary_layer 1
--- TVF:3 Calling Drc_SIZE @3 @0 BY 0.5 TRUNCATE 0.2
--- TVF:4 tvf::SETLAYER "@1 NOT @3"
```

Calling this function may generate many lines of output. It may be helpful to disable echoing of layer operations after a given number of loop iterations. This command will turn off echoing of layer operations.

```
tvf::echo_svrf 0; # turn off echoing to stdout
```

## Error Reporting

TVF errors are reported during runtime by the Tcl preprocessor. The error is reported like this:

```
*** Error in TVF script at line 7
<error details>
```

This is not the line number of the SVRF rule file, but a line number of a TVF Function or TVF layer operation initialization section. The SVRF compiler does not check the TVF code, so the compiler does not have a means of determining which line in the rule file has the Tcl error. However, using the error line number issued by the preprocessor, you can trace the exact line of TVF code where the error is located.

The line number reported by this error is with respect to the non-blank lines of a given TVF script. Each script has its lines sequentially numbered, starting from 1. The text of the error message should give you some idea which TVF Function generated the error. With that knowledge, you can count the non-blank lines of the script until you reach the line number shown in the error message. Or you can simply search your rule file for text matching the error text.

For runtime errors involving layer operation calls, if you have called “tvf::echo\_svrf 1”, as shown in the previous section, you can determine where an error occurs by calling a Tcl proc to set a traceback line number. You can do so in a TVF Function to get a trace for the output line. Here is an example of setting the trace line:

```
tvf::set_traceback_line 7; # layer operation trace
```

You will then see output in the run transcript like this:

```
--- TVF:1 tvf::SETLAYER one_not_two = @0 NOT @1
--- TVF:2 Calling Drc_new_edge_collection 1 keep
--- TVF:3 Calling Drc_NOT @3 @0 @1
--- TVF:4 Calling Drc_object_count @3
--- TVF:5 tvf::OUTLAYER COPY @3
--- TVF:6 Calling Drc_COPY @2 @3
--- TVF:7 Calling Drc_clear_edge_collection 0x2735d30 1
    Traceback forced at TVF layer operation number 7
2: traceLayerOperation Calling Drc_clear_edge_collection 0x2735d30 1
1: tvf::epilog
```

Here you can see where the traceback was forced and the operations that were performed.

The following command prints a trace of all TVF layer operation calls during runtime:

```
tvf::print_call_stack 1; # trace all layer operation calls
```

Using this command can generate many lines of output; setting “tvf::print\_call\_stack 0” after a given number of loop iterations may be helpful in reducing unwanted traces. The output looks like this:

```
--- TVF:1 tvf::SETLAYER one_not_two = @0 NOT @1
    Traceback forced at TVF layer operation number 0
2: traceLayerOperation {tvf::SETLAYER one_not_two = @0 NOT @1}
1: tvf::SETLAYER one_not_two = {@0 NOT @1}
--- TVF:2 Calling Drc_new_edge_collection 1 keep
    Traceback forced at TVF layer operation number 0
7: traceLayerOperation Calling Drc_new_edge_collection 1 keep
6: getNewLayer type_1
5: NOT @0 @1
4: LeafLayerOperation { @0 NOT @1}
3: parseImplicitLayerDefinition { @0 NOT @1} false top-level
2: tvf::parseLayerDefinition { @0 NOT @1}
```

```
1: tvf::SETLAYER one_not_two = {@0 NOT @1}
```

## Namespace Importation

All TVF commands begin with the tvf:: prefix. In practice, starting every line with tvf:: can become tedious. You can import available commands from the TVF namespace using the following Tcl command:

```
namespace import tvf::<command_name> [tvf::<command_name> ...]
```

Command names imported using this structure do not need to be preceded by “tvf::” within the script. You can import a TVF command only once in a script using this command.

For example, you can do the following:

```
namespace import tvf::SETLAYER tvf::OUTLAYER
```

This would allow you to write previously-shown TVF commands like this:

```
SETLAYER POLY1 = "poly NOT diff"
OUTLAYER "COPY $extl_in"
```

Note the absence of the tvf:: namespace prefix.

Most commands in the tvf:: namespace have both upper- and lowercase versions. If you want to import both, you must import them separately. For commands that are mixed case, you must import them as shown.

A recommended list of compile-time commands for namespace importation are these:

**Table 6-8. Compile-Time Commands for Namespace Importation**

TVF command	Summary
tvf::COMMENT	Denotes a rule file comment.
tvf::SETLAYER	Denotes a layer definition.
tvf::OUTLAYER	Denotes rule check output.
tvf::RULECHECK	Denotes a nmDRC rule check.
tvf::VERBATIM	Denotes verbatim SVRF syntax.
tvf:://	Denotes a rule file comment.
tvf::@	Denotes a rule check (checktext) comment.

A recommended list of TVF runtime commands for namespace importation are these:

**Table 6-9. Runtime Commands for Namespace Importation**

TVF command	Summary
tvf::DELETE_LAYER	Deletes a layer handle.

**Table 6-9. Runtime Commands for Namespace Importation**

TVF command	Summary
tvf::GET_LAYER_ARGS	Retrieves arguments passed from a TVF layer operation.
tvf::GET_LAYER_EXTENT	Retrieves the extent of a layer.
tvf::IS_LAYER_EMPTY	Determines whether a layer is empty.
tvf::OUTLAYER	Denotes TVF layer operation output.
tvf::SETLAYER	Denotes a layer definition.
tvf::svrf_var	Denotes access to an SVRF Variable statement name.
tvf::sys_var	Denotes access to a system variable.

The namespace import command allows you to use the “\*” wildcard to match zero or more characters. For example:

```
namespace import tvf::L*
```

This command imports all TVF command names beginning with L.

---

**Note**

 You should use namespace importation, especially use of wildcards, with care. You can cause unexpected collisions between imported command names and other Tcl procs in your rule file.

---

## Precautions for the Compile-Time Environment

Although the namespace import command makes it easy to convert a large number of SVRF statements to compile-time TVF syntax, converting large numbers of SVRF statements without careful examination exposes you to potential Tcl/SVRF syntax inconsistencies. Here are examples:

- The arguments of the TVF comment commands tvf::@, tvf:://, and tvf::COMMENT are evaluated by Tcl. If the arguments to these commands contain brackets ([...]), then the contents of the brackets are executed as a command. This usually results in the error message: invalid command name. Worse, however, is the possibility that the brackets enclose a valid command that causes an unwanted side effect.

Undefined variables in the comment’s argument fields also give an error message: can’t read <variable>, no such variable.

- Single quotes are not special characters in Tcl and are passed through to the SVRF output; however, the blanks between the single quotes are not protected. If such blanks are important to maintain, they should be protected by double quotes or braces.

- In TVF, a line beginning with a series of three or more slashes (///) generates an undefined command error. Slashes are not special characters—Tcl interprets this string as a command that it cannot find. In TVF, there is a defined comment command // but it must be followed by a blank space to be recognized.

# TVF Command Reference

[Table 6-10](#) describes the available TVF commands.

**Table 6-10. TVF Commands**

Command Name	Description
<code>tvf:// comment</code>	<p>Specifies a single-line user-supplied comment that does not affect code execution.</p> <p>Application: compile-time only.</p>
<code>tvf:@ comment</code>	<p>Specifies a single-line user-supplied comment that does not affect code execution. Must appear inside a rule check. Can also be specified with <code>tvf:COMMENT</code>. For example,</p> <p><code>tvf:@ A rule check comment.</code></p> <p>Application: compile-time only.</p>
<code>tvf:block_transcript</code>	<p>Disables the echoing of the rule file selected with the <code>-E</code> command-line option and to disables the transcript output produced by SVRF commands.</p> <p>Application: compile-time only.</p>
<code>tvf:COMMENT comment</code>	<p>Identical to <code>tvf:@ comment</code>.</p> <p>Application: compile-time only.</p>
<code>tvf:COPY layer</code>	<p>Identical to the SVRF Copy operation. This is the only TVF command mapped to an SVRF layer operation.</p> <p>Application: compile-time only.</p>
<code>tvf:DELETE_LAYER \$layer</code>	<p>Deletes the layer specified by <code>\$layer</code>. Subsequent references to <code>\$layer</code> return an error. This command is useful for reducing peak memory usage by removing temporary layers from the run.</p> <p>Application: runtime only.</p>
<code>tvf:echo_svrf { 0   1 } [file_id]</code>	<p>When enabled (set to 1), echoes to the transcript SVRF created from translating TVF. This command is disabled (set to 0) by default.</p> <p>By default the echo is sent to stdout. The output is sent to a file when the file handle <code>file_id</code> is included.</p> <p>Application: compile-time only.</p>

**Table 6-10. TVF Commands (cont.)**

<b>Command Name</b>	<b>Description</b>
<code>tvf::echo_to_svrf_file <i>string</i></code>	Writes a string directly to the output SVRF file. This command is used only in pre-processor mode.  Application: compile-time only.
<code>tvf::epilog</code>	An internal procedure that automatically cleans up temporary and deleted layers.  Application: runtime only.
<code>tvf::get_calibre_version</code>	Returns the Calibre version you are running.
<code>tvf::get_tvf_arg</code>	Returns the argument specified with the <code>-tvfarg</code> command-line option. For example, if  <code>calibre -tvfarg my_argument</code>  is used at the command line, then <i>my_argument</i> can be accessed in compile-time TVF code with:  <code>set tvf_arg [tvf::get_tvf_arg]</code>  Application: compile-time only.
<code>tvf::GET_LAYER_ARGS <i>layer1</i> [<i>layerN</i> ...] [<i>args</i>]</code>	Imports the layers passed to a TVF Function by a TVF layer operation, and sets up the layer handles \$layer1, \$layer2, and so on. The number of imported layer arguments must match the number of layer arguments passed by the TVF layer operation.  The optional args parameter is a literal name used to reference a list of layer arguments passed from a TVF layer operation. If args is specified, it must be the final argument in this command.  This command must be used once in a TVF Function statement that is called by a TVF layer operation.  Application: runtime only.
<code>tvf::GET_LAYER_EXTENT \$layer</code>	Returns the bounding box of the extent of the layer as a list in the format {x1 y1 x2 y2}.  Application: runtime only.
<code>tvf::IS_LAYER_EMPTY \$layer</code>	Used to test for an empty layer. Returns true (1) if \$layer is empty.  Application: runtime only.

**Table 6-10. TVF Commands (cont.)**

Command Name	Description
tvf::is_transcript_blocked	<p>Checks if the calibre transcript output is blocked. Returns a value of 1 if the output to calibre transcript is currently blocked or 0 otherwise.</p> <p>Application: compile-time only.</p>
tvf::OUTLAYER <i>arguments</i>	<p>OUTLAYER provides a means of doing layer operations in both runtime and compile-time TVF, where <i>arguments</i> is a valid layer operation. See <a href="#">Table 6-6</a> for restrictions on the layer operations in runtime operation.</p> <p>The OUTLAYER command arguments may be enclosed in double quotes or braces. Both of these grouping elements may appear in the same group of arguments. Double quotes (" ") allow for variable substitution in a list of arguments, whereas braces ({ }) pass their contents as a list with no variable substitution.</p> <p><b>Compile-time operation</b> — Sends the results of <i>arguments</i> to the DRC results database. OUTLAYER is used within the tvf::RULECHECK command. See <a href="#">Rule Checks</a> for several examples.</p> <p><b>Runtime operation</b> — Executes the <i>arguments</i> and returns the layer to the TVF operation that called the TVF Function. The corresponding TVF Function statement must have exactly one tvf::OUTLAYER invocation, and the output layer type must match the output type specified for the TVF Function. See <a href="#">Example 6-9</a>.</p> <p>Application: runtime and compile-time.</p>

**Table 6-10. TVF Commands (cont.)**

Command Name	Description
tvf::print_call_stack { 0   1 }	<p>Prints a trace of all TVF layer operation calls during runtime:</p> <pre>tvf::print_call_stack 1; # trace all layer operation calls</pre> <p>Using this command can generate many lines of output; setting “tvf::print_call_stack 0” after a given number of loop iterations may be helpful in reducing unwanted traces. The output looks like this:</p> <pre>--- TVF:1 tvf::SETLAYER one_not_two = @0 NOT @1 Traceback forced at TVF layer operation number 0 2: traceLayerOperation {tvf::SETLAYER one_not_two = @0 NOT @1} 1: tvf::SETLAYER one_not_two = {@0 NOT @1} --- TVF:2 Calling Drc_new_edge_collection 1 keep Traceback forced at TVF layer operation number 0 7: traceLayerOperation Calling Drc_new_edge_collection 1 keep 6: getNewLayer type_1 5: NOT @0 @1 4: LeafLayerOperation { @0 NOT @1} 3: parseImplicitLayerDefinition { @0 NOT @1} false top-level 2: tvf::parseLayerDefinition { @0 NOT @1} 1: tvf::SETLAYER one_not_two = {@0 NOT @1}</pre> <p>Application: runtime only.</p>

**Table 6-10. TVF Commands (cont.)**

Command Name	Description
tvf::RULECHECK { <i>rulecheck</i> }	<p>Denotes a nmDRC rule check. See <a href="#">Rule Checks</a> for discussion of the valid TVF commands in the <i>rulecheck</i> block.</p> <p>Rule checks in SVRF have user-given names, so the SVRF syntax cannot be copied in TVF. Instead, this construct is handled by a tvf::RULECHECK command. For example, the following SVRF rule check:</p> <pre>my_rule {     layer1 = layer2 OR layer3     layer4 NOT layer 1 }</pre> <p>is coded like this in TVF:</p> <pre>tvf::RULECHECK my_rule {     tvf::SETLAYER layer1 = {layer2 OR     layer3}     tvf::OUTLAYER {layer4 NOT layer1} }</pre> <p>Application: compile-time only.</p>
tvf::set_disclaimer <i>disclaimer</i>	<p>Writes a formatted header, specified with the <i>disclaimer</i> string, to the top of the SVRF output file and replaces the default header. This command, if used, must be the first command specified in your compile-time TVF file.</p> <p>Application: compile-time, pre-processor mode only.</p>
tvf::set_rule_check_indentation <i>indentation</i>	<p>Controls indentation of each rule check printed in the output SVRF file (the default is a tab). The indentation parameter is a string that is pre-pended to each line. If the indentation is an empty string (""), rule check lines are not indented.</p> <p>Application: compile-time, pre-processor mode only.</p>

**Table 6-10. TVF Commands (cont.)**

Command Name	Description
tvf::set_traceback_line <i>line_number</i>	<p>Used to trace SVRF syntax errors at <i>line_number</i> to line numbers in your TVF rule file.</p> <p>For example, if the SVRF compiler reports this error:</p> <pre>ERROR: Error INP3 on line 52 of rules.tvf</pre> <p>You add this line to your TVF code:</p> <pre>tvf::set_traceback_line 52; # trace line 52</pre> <p>This instructs the Tcl preprocessor to issue an error message when the output line number is equal to 52. When you run Calibre again on the TVF file that contains this command, the TVF line number is reported as shown here:</p> <pre>ERROR: Error TVF1 on line 79 of rules.tvf - TVF preprocessor error: Traceback forced at SVRF output line number 52</pre> <p>This tells you the error is on line 79 of the TVF file.</p> <p>Application: compile-time and runtime.</p>

**Table 6-10. TVF Commands (cont.)**

Command Name	Description
tvf::SETLAYER <i>layer_name</i> = <i>arguments</i>	<p>Executes the <i>arguments</i> (which are layer operations) and stores the output in the <i>layer_name</i> handle. Used for layer derivation in both compile-time and runtime.</p> <p>See <a href="#">Table 6-6</a> and the section <a href="#">Layer Operations</a> for restrictions on the layer operations and syntax in runtime operation.</p> <p>The SETLAYER command <i>arguments</i> may be enclosed in double quotes or braces. Both of these grouping elements may appear in the same group of arguments. Double quotes (" ") allow for variable substitution in a list of arguments, whereas braces ({ }) pass their contents as a list with no variable substitution.</p> <p>The layer operations specified to SETLAYER use the same syntax as corresponding SVRF layer operations, with some restrictions for runtime operation. Layer operations are passed as arguments to the SETLAYER command, and are then passed to the SVRF compiler. The return value from SETLAYER command is the name of the layer.</p> <pre>tvf::SETLAYER lout = ((L1 AND L2) OR "(\$L3 AND \$L4)") AND {(L5 OR L6)}</pre> <p>Application: runtime and compile-time. Also see <a href="#">SETLAYER Options in Compile-Time TVF</a>.</p>
tvf::svrf_var <i>variable_name</i>	<p>Allows you to pass SVRF Variable specification statements to a TVF Function.</p> <p>Any <i>variable_name</i> that is declared in a Variable statement in the rule file is accessible in a TVF Function using this construct.</p> <p>Application: runtime only.</p>
tvf::sys_var <i>variable</i>	<p>Accesses the value of a system variable. <a href="#">Table 6-7</a> contains a list of supported system variables. For example, the following code accesses the value of PRECISION:</p> <pre>set precision [tvf::sys_var PRECISION]</pre> <p>Application: runtime only.</p>

**Table 6-10. TVF Commands (cont.)**

<b>Command Name</b>	<b>Description</b>
tvf::unblock_transcript	<p>Enables echoing of the rule file selected with the -E command-line option and to enables the transcript output produced by SVRF commands.</p> <p>Application: compile-time only.</p>
tvf::VERBATIM { <i>verbatim_SVRF_code</i> }	<p>Compile-time TVF provides this command to pass in verbatim SVRF code.</p> <p>Whatever appears between the braces is passed directly to the SVRF compiler and must adhere to SVRF syntactical conventions. Any errors in the SVRF syntax generate a Calibre compiler error, as usual. You should use this command as much as possible in your compile-time TVF scripts to pass blocks of SVRF code that do not need to be changed to TVF syntax. This includes runtime TVF commands.</p> <p>The following example shows how a typical SVRF rule check can be passed as-is within the VERBATIM command:</p> <pre>tvf::VERBATIM {     //Minimum m1 width is 0.10     m1_width {         INT m1 &lt; .10 ABUT &lt; 90 SINGULAR     } }</pre> <p>Application: compile-time only.</p>

## Global Variables

These global variable commands work in compile-time TVF. They allow global variables to be shared between interpreters, encrypted and unencrypted. The following commands allow the user to manipulate global variables. If the optional index arguments are provided for the invocation of these commands, *variable\_name* is assumed to be the name of an array, while the index values determine the array's element. Additionally, the index values can optionally be

surrounded by parentheses and separated by commas. See the [TVF encrypt User's Manual](#) for more details.

**Table 6-11. Global Variables**

Command Name	Description
<code>tvf::exists_global_variable variable_name [(index1 index2)]</code>	Checks for the existence of a global variable and optional <i>index</i> values. Returns 1 if the variable exists, 0 otherwise.
<code>tvf::get_global_variable variable_name [(index1 index2)]</code>	Retrieves the value of a global TVF variable and optional index values. Returns the values of the global variable if it exists and returns ““ otherwise.
<code>tvf::set_global_variable variable_name [(index1 index2)] value</code>	Sets the value of a TVF global variable given its name and optional index values. The variable is created if it does not exist. Returns 1 on success, 0 otherwise.
<code>tvf::unset_global_variable variable_name [(index1 index2)]</code>	Clears the global variable and optional index values. Returns 1 if the global variable existed prior to calling this function, 0 otherwise.

## TVF Examples

### Example 1

The following compile-time TVF script shows how you can derive layers with TVF and simplify derivations:

```
#! tvf
namespace import tvf::VERBATIM tvf::SETLAYER

VERBATIM {
    LAYOUT PATH "test.gds"
    LAYOUT PRIMARY "test"
    LAYOUT SYSTEM GDSII
    DRC RESULTS DATABASE
    drc.test.db
    DRC SUMMARY REPORT drc.summary.rpt
    LAYER OD 1
    LAYER P1 2
    LAYER P2 3
    LAYER CO 4
    LAYER M1 5
    LAYER V1 6
    LAYER M2 7
    LAYER V2 8
    LAYER M3 9
    LAYER V3 10
    LAYER M4 11
    LAYER V4 13
    LAYER M5 14
    LAYER V5 15
    LAYER M6 16
```

```

LAYER V6 17
LAYER M7 18
LAYER V7 19
LAYER M8 20
LAYER V8 21
LAYER M9 22
LAYER BOUNDARY 100
}

foreach {lay} { OD PO1 PO2 CO M1 V1 M2 V2 M3 V3 M4 V4 M5 V5 M6 V6 M7 V7 M8
V8 M9 } {
    SETLAYER ${lay}_logo = ${lay} INSIDE BOUNDARY
    SETLAYER ${lay}_RAM = (INSIDE CELL ${lay} RAM) OUTSIDE BOUNDARY
    SETLAYER ${lay}_NRAM = (NOT INSIDE CELL ${lay} RAM) OUTSIDE BOUNDARY
    SETLAYER ${lay}_o = ${lay} OUTSIDE BOUNDARY\n
}

```

## Example 2

The following compile-time TVF script shows how you can pass an environment variable into a layer derivation. To run this script, you must first set the environment variable “mycell” to the cellname “bit”.

```

#! tvf
namespace import tvf::SETLAYER tvf::VERBATIM tvf::RULECHECK tvf::OUTLAYER
tvf:@ tvf::VARIABLE

SETLAYER cell_extent = "EXTENT CELL $env(mycell) ORIGINAL"

VERBATIM {
    LAYOUT PRIMARY "TOP"
    LAYOUT PATH      "TOP.gds"

    LAYOUT SYSTEM   GDSII

    DRC SUMMARY REPORT      "drc.report"
    DRC RESULTS DATABASE "drc.rve" ASCII

    MASK RESULTS DATABASE none
    DRC MAXIMUM VERTEX 199
    PRECISION 1000
    RESOLUTION 1
    UNIT LENGTH U

    DRC MAXIMUM RESULTS ALL

    FLAG SKEW YES
    FLAG OFFGRID YES
    FLAG ACUTE YES
    FLAG NONSIMPLE YES

    LAYOUT TOP LAYER met1 via1 met2 via2 met3 via3 met4 via4 met5

    LAYER nwel    2      // N-Well
    LAYER pdif    11     // Thin oxide w/PP
    LAYER ndif    12     // Thin oxide w/NP
}
```

```
LAYER poly 13      // Poly
LAYER cont 15      // Contact
LAYER vial 17      // Via 1
LAYER via2 27      // Via 2
LAYER via3 29      // Via 3
LAYER via4 32      // Via 4

LAYER met1 16      // Metal 1
LAYER met2 18      // Metal 2
LAYER met3 28      // Metal 3
LAYER met4 31      // Metal 4
LAYER met5 33      // Metal 5

psub = EXTENT

R1 {@ Extent cell
    COPY cell_extent
}
} ;# VERBATIM
```

### Example 3

The following compile-time TVF script contains a procedure that accepts original or derived layer names as input and generates rules that copy each layer.

```
#! tvf

tvf::VERBATIM {
INCLUDE "./drc_rules"
}

proc copy_layers { layers } {
    foreach lay $layers {
        tvf::DRC SELECT CHECK COPY_LAY_$lay
        tvf::RULECHECK COPY_LAY_$lay {
            eval {tvf::@} Copy of layer $lay
            tvf::OUTLAYER "COPY $lay"
        }
    }
}

copy_layers { gate pmos nmos }
```

This code generates the following output:

```
INCLUDE "./drc_rules"

DRC SELECT CHECK COPY_LAY_gate
COPY_LAY_gate {
    @ Copy of layer gate
    COPY gate
}
DRC SELECT CHECK COPY_LAY_pmos
COPY_LAY_pmos {
```

```

        @ Copy of layer pmos
        COPY pmos
    }
DRC SELECT CHECK COPY_LAY_nmos
COPY_LAY_nmos {
    @ Copy of layer nmos
    COPY nmos
}

```

#### Example 4

The following compile-time TVF script contains a procedure that accepts original GDS layer numbers as input and generates both LAYER statements and rules to copy the original layers out:

```

#! tvf

tvf::VERBATIM {
LAYOUT PATH "./test.gds"
LAYOUT PRIMARY "top"
LAYOUT SYSTEM GDSII

PRECISION 1000

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "./test.rve" ASCII
}

proc simple_layer { layers } {
    foreach lay $layers {
        set lname sl_$lay
        tvf::LAYER $lname $lay
        tvf::RULECHECK COPY_LAY_$lay { eval {tvf::@} Copy of layer $lay
        tvf::OUTLAYER "COPY $lname"
    }
}
simple_layer { 1 2 3 4 5 }

```

This code generates the following output:

```

LAYOUT PATH "./test.gds"
LAYOUT PRIMARY "top"
LAYOUT SYSTEM GDSII

PRECISION 1000

DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "./test.rve" ASCII

LAYER sl_1 1
COPY_LAY_1 {@ Copy of layer 1
            COPY sl_1
}
LAYER sl_2 2
COPY_LAY_2 {@ Copy of layer 2
            COPY sl_2
}

```

```
}

LAYER sl_3 3
COPY_LAY_3 {@ Copy of layer 3
    COPY sl_3
}
LAYER sl_4 4
COPY_LAY_4 {@ Copy of layer 4
    COPY sl_4
}
LAYER sl_5 5
COPY_LAY_5 {@ Copy of layer 5
    COPY sl_5
}
```

# Chapter 7

## Error Messages

---

During compilation of a rule file, any syntax and semantic errors generated are reported in the format (unless noted otherwise):

```
Error <enum> on line <lnum> of <fname> -- <message>
```

where:

- <enum> is the unique error number of the compiler error and is reported by mnemonic in both the error message and in the following tables.
- <fname> is the name of the rule file in which the error occurred. <fname> differs from the name of the primary rule file only if other rule files are included within it (via the Include statement).
- <lnum> is the line number within <fname> on which the error occurred.
- <message> is a message describing the error.

At most, one error is reported during each compilation attempt.

Syntax and compilation errors are not generally generated in the order in which they may seem to occur because there is no fixed ordering of statements and operations within the rule file. For example, a compilation error can be reported at position X in the rule file even though there is an error prior to position X. Thus, compilation is sequenced by priority and not by statement and operation order.

## Compilation Errors

Tables 7-1 through 7-55 list and describe the messages that result from rule file compilation errors.

## Continuous nmDRC Errors

**Table 7-1. Continuous nmDRC Errors**

Error #	Explanation
CDRC1	A rule check statement defined in the continuous_drc rule check group contains more than one output-producing operation. The name of the erroneous rule check statement appears in the error message.
CDRC2	The output-producing operation of a rule check statement defined in the continuous_drc rule check group is not an error-directed dimensional check operation. The name of the erroneous rule check statement appears in the error message.
CDRC3	The input layer(s) for the output-producing operation of a rule check statement defined in the continuous_drc rule check group are not original layers. The name of the erroneous rule check statement appears in the error message.

## LVS Cell List Statement Errors

**Table 7-2. LVS Cell List Statement Errors**

Error #	Explanation
CEL1	An LVS Cell List statement has a missing or invalid cell list name.
CEL2	There is a duplicate LVS Cell List statement.
CEL3	An LVS Cell List statement is missing a cell name.
CEL4	An LVS Cell List statement has an invalid cell name.
CEL5	A cell name is duplicated in one or more LVS Cell List statements.

## Constraint Errors

**Table 7-3. Constraint Errors**

Error #	Explanation
CNS1	A constraint is improperly formed (the constraint token is at the end of the operation, the keywords are within the constraint in place of numbers or variables, an interval constraint is not properly specified, or variables appear in the constraint in a context where they are not allowed).
CNS2	The range of a constraint is invalid for all operations and statements in which it appears. This occurs when an interval constraint variable a is not less than b.

**Table 7-3. Constraint Errors (cont.)**

Error #	Explanation
CNS3	The range of a constraint is invalid for the particular operation in which it appears, or for the particular secondary keyword that it modifies. The primary keyword of the operation or the name of the secondary keyword appears in the error message. This occurs when: <ol style="list-style-type: none"><li>1. The constraint value(s) for an Angle operation are not in the range [0,90] or the constraint is <math>\{x \mid x &gt; 90\}</math>.</li><li>2. The constraint for any dimensional check operation is <math>&gt; a</math>, <math>\geq a</math>, or <math>\neq a</math>.</li><li>3. The constraint value(s) for ABUT are not in the range [0,180], the constraint is <math>\{x \mid x &gt; 180\}</math>, or the constraint interval includes 180.</li><li>4. The constraint for ANGLED does not include 0, 1, or 2 in its range.</li><li>5. The constraint for CORNER TO CORNER is not <math>\equiv 45</math> or <math>\neq 45</math>.</li><li>6. The constraint for WIDTH in the Opclineend operation is not <math>&gt; a</math>, <math>\geq a</math>, or <math>= a</math>.</li></ol>
CNS4	The constraint for the ABUT parameter for a one-layer dimensional check is $\{x \mid x = 0\}$ or $\{x \mid x \leq 0\}$ .
CNS5	A dimensional check operation, polygon measurement operation, Density operation, Net Area Ratio operation, Net Area Ratio ACCUMULATE operation, With Neighbor operation, edge measurement operation, or the (Not) Rectangle operation with MEASURE EXTENTS lacks a constraint. The name of the erroneous operation appears in the error message.
CNS6	An operation has a superfluous constraint specification. The name of the erroneous operation appears in the error message.

## DRC Check Map Statement Errors

**Table 7-4. DRC Check Map Statement Errors**

Error #	Explanation
DCM1	A DRC Check Map specification statement is missing its nmDRC rule check name.
DCM2	The layer number parameter of a DRC Check Map specification statement is negative, is non-integral, or is greater than 65535 (the maximum allowable original layer number). The rule check name of the erroneous statement appears in the error message.
DCM3	The datatype parameter of a DRC Check Map specification statement is not a number, is negative, is non-integral, or is greater than 65535 (the maximum allowable datatype value). The rule check name of the erroneous statement appears in the error message.
DCM4	The rule check name in a DRC Check Map specification statement is not that of a nmDRC rule check or rule check group in the rule file. The rule check name of the erroneous statement appears in the error message.
DCM5	The rule check name in a DRC Check Map specification statement duplicates that of another DRC Check Map specification statement in the rule file and 1. both have unspecified <filename> parameters, or 2. one has an unspecified <filename> parameter and the other's <filename> parameter matches that specified in the DRC Results Database Specification statement, 3. both <filename> parameters are specified and they match. The rule check name of the errant statement is placed into the message.
DCM6	The cell name following the AREF keyword in a DRC Check Map specification statement is repeated in the same DRC Check Map specification statement; appears in a different DRC Check Map specification statement with unequal layers, datatypes, widths, lengths' or SUBSTITUTE polygons; or has its width and length parameters equal to those (in either order) of another AREF cell name in the same DRC Check Map specification statement. The cell name of the erroneous AREF keyword appears in the message.
DCM7	A DRC Check Map specification statement has a <filename> parameter specified and one of the following is true: <ul style="list-style-type: none"> <li>• The &lt;filename&gt; matches that of the DRC Results Database specification statement but the types do not agree.</li> <li>• The &lt;filename&gt; matches that of another DRC Check Map specification statement but the types do not agree.</li> </ul> The rule check name of the erroneous statement appears in the message.
DCM8	A DRC Check Map specification statement does not have a <filename> parameter specified and its type does not match the global type of the DRC Results Database specification statement. The rule check name of the erroneous statement appears in the message.

**Table 7-4. DRC Check Map Statement Errors (cont.)**

Error #	Explanation
DCM9	The MAXIMUM RESULTS keyword in a DRC Check Map specification statement has one of the following errors: <ul style="list-style-type: none"> <li>• Is missing its parameter.</li> <li>• Has a parameter which is not numeric or ALL.</li> <li>• Has a numeric parameter which is negative.</li> <li>• Is not an integer.</li> </ul> The rule check name of the erroneous statement appears in the message.
DCM10	A DRC Check Map specification statement for a nmDRC rule check R has a MAXIMUM RESULTS keyword specified and either: <ol style="list-style-type: none"> <li>1. Is not specified in all DRC Check Map specification statements for R.</li> <li>2. Is specified with a different value in some DRC Check Map specification statement for R.</li> </ol> The name of the erroneous rule check (R) appears in the message.
DCM11	The MAXIMUM VERTICES keyword in a DRC Check Map specification statement is: <ol style="list-style-type: none"> <li>1. missing its parameter,</li> <li>2. has a non-numeric parameter other than ALL,</li> <li>3. has a numeric parameter less than 4, or</li> <li>4. is not an integer.</li> </ol> The rule check name of the erroneous statement is in the message.
DCM12	A DRC Check Map statement for a nmDRC Rule Check R has a MAXIMUM VERTICES keyword specified and, either it is not specified for all DRC Check Map statements for R, or it has different values in some DRC Check Map statement. The rule check name of the erroneous statement is in the message.
DCM13	The coordinate list following the SUBSTITUTE keyword in a DRC Check Map AREF specification is too short (less than four numerics), or has an odd number of numerics, or is an invalid orthogonal rectangle, or represents a self-intersecting polygon. The cell name of the offending aref keyword is placed into the message.
DCM14	A DRC results database has an inconsistent PSEUDO, USER, or USER MERGED state. This is caused by either multiple DRC Check Map specification statements mapping to the same DRC results database but with a different PSEUDO, USER, or USER MERGED state, or a DRC Check Map specification statement mapping to the global DRC results database but with a different PSEUDO, USER, or USER MERGED state. The rule check name of the offending statement is placed into the message.
DCM15	The AREF keyword is specified in a DRC Check Map specification statement whose (default) DRC results database type is not GDSII or OASIS.
DCM16	The AUTOREF keyword is specified in a DRC Check Map specification statement whose (default) DRC results database type is not GDSII.

**Table 7-4. DRC Check Map Statement Errors (cont.)**

Error #	Explanation
DCM17	A DRC results database has an inconsistent AUTOREF prefix specification. This is caused by at least two DRC Check Map specification statements mapping to the same DRC results database such that one has an AUTOREF prefix specified and the other does not, or both do and they are unequal. The rule check name of the offending statement is in the message.
DCM18	The APPEND, PREFIX, or TEXTTAG keywords may only be specified in the DRC Check Map specification statement if the DRC results database type is GDSII or OASIS.
DCM19	A DRC results database has an inconsistent APPEND state. This is caused by 1) multiple DRC Check Map specification statements mapping to the same DRC results database, but with different APPEND <name> parameters (including one with and one without APPEND), or 2) a DRC Check Map specification statement mapping to the global DRC Results Database, but with different APPEND <name> parameters (including one with and one without APPEND). The same cases hold for the PREFIX keyword. The rule check name of the offending statement is placed into the message.
DCM20	The PROPERTIES keyword is specified in a DRC Check Map specification statement but the DRC results database type of the statement is not GDS or OASIS. The rule check name of the offending statement is placed into the message.
DCM21	The PROPERTIES keyword in a DRC Check Map specification statement may not be specified if the AREF or AUTOREF keyword is also specified. The rule check name of the offending statement is placed into the message.

## Pre-Processor Directive Errors

**Table 7-5. Pre-Processor Directive Errors**

Error #	Explanation
DEF1	A #DEFINE, #UNDEFINE, #IFDEF, or #IFNDEF pre-processor directive is missing its operand. The name of the offending pre-processor directive is placed into the message.
DEF2	A #DEFINE, #IFDEF, or #IFNDEF pre-processor directive has more than two operands on the same line, or a #UNDEFINE, pre-processor directive has more than one operand on the same line, or a #ELSE or #ENDIF pre-processor directive has more than zero operands on the same line. The name of the offending pre-processor directive is placed into the message.
DEF3	A #ELSE preprocessor directive is not preceded by a #IFDEF or #IFNDEF preprocessor directive, or is preceded by another #ELSE preprocessor directive.
DEF4	A #ENDIF preprocessor directive is not preceded by a #IFDEF, #IFNDEF, or #ELSE preprocessor directive.
DEF5	A conditional lacks a #ENDIF preprocessor directive.

## Density Operation Errors

**Table 7-6. Density Operation Errors**

Error #	Explanation
DEN1	The STEP parameter in a Density operation may not be specified unless WINDOW is also specified.
DEN2	Both WINDOW and STEP are specified in a Density operation and a STEP value exceeds the corresponding WINDOW value or does not evenly divide it. The errant STEP value is placed into the message.
DEN3	A Density or Density Convolve operation has a superfluous expression specification().
DEN4	Bad syntax in a Density expression. The object (number, keyword, and so on.) where the illegal syntax was detected is placed into the message.
DEN5	The layer parameter to an AREA( <i>layer</i> ) function in a Density expression is not an input layer in the same operation. The name of the errant layer is placed into the message.
DEN6	The CORNER keyword is specified in a Density operation without the GRADIENT keyword. The name of the errant keyword is placed into the message.
DEN7	The CENTERED, BY EXTENT, or BY POLYGON keyword is specified in a Density operation without the INSIDE OF LAYER keyword. The name of the errant keyword is placed into the message.

**Table 7-6. Density Operation Errors**

Error #	Explanation
DEN8	The ABSOLUTE or RELATIVE keyword is specified in a Density operation without the GRADIENT, MAGNITUDE, or keyword. The name of the offending keyword is placed into the message.
DEN9	The Density Convolve operation has more than one input layer.
DEN10	A secondary keyword valid in the Density operation is not allowed in the Density Convolve operation. The name of the offending keyword is placed into the message.
DEN11	The WINDOW keyword of the Density operation may have only one parameter when used in Density Convolve.
DEN12	The Density Convolve operation must have a FILE parameter.
DEN13	The file parameter of a Density Convolve operation must be followed by a file name or by an inlined file.
DEN14	The Density operation contains the MAG or keyword, which requires that either the RDB or RDB ONLY keyword must also be present.
DEN15	The COMBINE keyword may not be specified in a Density operation if any of GRADIENT, MAGNITUDE, CENTERS, BY POLYGON, or BY RECTANGLE are also specified.

## Device Definition Errors

**Table 7-7. Device Definition Errors**

Error #	Explanation
DEV1	A Device operation is missing its element name argument.
DEV2	A Device operation is missing its device layer argument.
DEV3	A Device operation's model or pin name, or pin swap argument is improperly formed. That is, the right parenthesis is not seen, a keyword or number is encountered before the right parenthesis, or the specification is empty.
DEV4	A Device operation's pin name argument directly follows the device layer argument. The name of the device layer appears in the error message.
DEV5	The Device operation has fewer than three pin layers specified if the element name is MN, MP, MD, ME, or Q; has fewer than two pin layers specified if the element name is C, D, or R; has no pin layers specified if the element name is that of a generic Device operation. The element name of the erroneous device definition appears in the error message.
DEV6	A Device operation's pin swap parameter contains only one pin name within the group. The pin name appears in the error message.
DEV7	A pin layer in a generic Device operation, any pin layer past the fourth in a device recognition operation with element name MN, MP, MD, ME, or Q, or any pin layer past the third in a Device operation with element name C, D, or R, is missing a pin name parameter. The name of the pin layer appears in the error message.
DEV8	The pin name specified for a pin layer in a nongeneric Device operation is invalid given the element name and pin layer ordinal. The erroneous pin name appears in the error message.
DEV9	A pin name within a Device operation appears after more than one pin layer. The erroneous pin name appears in the error message.
DEV10	An auxiliary layer specification within a Device operation is improperly formed. That is, the right angle bracket is not seen, a keyword or number is encountered before the right angle bracket, or the specification is empty.
DEV11	A name within a pin swap group of a Device operation is not the pin name for any pin layer in the operation. The erroneous pin name appears in the error message.
DEV12	A pin name in a Device operation is within multiple pin swap groups in the operation or appears more than once in the same pin swap group. The erroneous pin name appears in the error message.
DEV13	Deprecated.

**Table 7-7. Device Definition Errors (cont.)**

Error #	Explanation
DEV14	A pin swap group containing the pin name for a given pin layer does not contain all of the pin names for that layer within a Device operation. The name of the erroneous pin layer appears in the error message.
DEV15	The device layer indicated in the error message is used as a pin layer more than once in a Device operation.
DEV16	Two device recognition operations with the same element name, same optional model name, and same number of pin layers have a different set of pin names. The name of the erroneous device layer appears in the error message.
DEV17	Two device recognition operations with the same element name and same number of pin layers have a different set of pin names. The errant element name is placed into the message.
DEV18	Two device recognition operations with the same element name and model name and same number of pin layers have different swappability conditions for their pin layers, as specified by the pin swap groups in each of the two definitions. The errant element(model) name is placed into the message.
DEV19	An auxiliary layer in a device recognition operation coincides with the device layer. The name of the erroneous auxiliary layer appears in the message.
DEV20	An auxiliary layer in a device recognition operation appears more than once in the same operation. The name of the erroneous auxiliary layer appears in the message.
DEV21	An auxiliary layer in a device recognition operation coincides with a pin layer in some device recognition operation having the same device layer. The name of the erroneous auxiliary layer appears in the message.
DEV22	A device recognition operation has a superfluous property specification ([]).
DEV23	Two device recognition operations with the same device layer have different pin recognition algorithms as specified by the secondary keywords BY NET and BY SHAPE, or their defaults. The name of the erroneous device layer appears in the message.
DEV24	A device recognition operation device layer, pin layer, or auxiliary layer is derived from an ERC operation, namely Pathchk or Device Layer.
DEV25	Two device recognition operations with the same device layer have different Text Model Layers specified. The name of the erroneous device layer appears in the message.
DEV26	A context layer is repeated in a Device SIGNATURE statement.
DEV27	A Device SIGNATURE statement is missing a reference layer.
DEV28	A Device SIGNATURE statement is missing a context layer.

**Table 7-7. Device Definition Errors (cont.)**

Error #	Explanation
DEV29	The Device SIGNATURE keyword is present but the statement lacks context or reference layers.
DEV30	A Device signature string is invalid.
DEV31	A Device SIGNATURE statement is missing a reference layer after a signature string.
DEV32	A Device SIGNATURE statement is missing at least one context layer following the reference layer.
DEV33	A Device SIGNATURE reference layer is associated with more than one context layer list in all of current Device definitions.
DEV34	A Device SIGNATURE reference layer appears in a context layer list.

## DFM Operation Errors

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM1	The STEP keyword may not be specified in DFM Analyze unless WINDOW is specified and NUMWIN is not used.
DFM2	Both WINDOW and STEP are specified in a DFM Analyze operation and a STEP value exceeds the corresponding WINDOW value, or does not evenly divide it. The errant STEP value is placed into the message.
DFM3	A DFM Analyze operation has a superfluous expression specification.
DFM4	Bad syntax in a DFM Analyze expression. The object where the illegal syntax was detected is placed into the message.
DFM5	The layer parameter to an AREA( <i>layer</i> ), PERIMETER( <i>layer</i> ), LENGTH( <i>layer</i> ), or COUNT( <i>layer</i> ) function in a DFM Analyze expression is not an input layer in the same operation. The name of the errant layer is placed into the message.
DFM6	The parameter to a LENGTH( <i>layer</i> ) function in a DFM Analyze expression may only be a derived edge layer. The name of the errant layer is placed into the message.
DFM7	The parameter to an AREA( <i>layer</i> ) or PERIMETER( <i>layer</i> ) function in a DFM Analyze expression may only be an original or derived polygon layer. The name of the errant layer is placed into the message.
DFM8	The layer parameter to an AREA( <i>layer</i> ), PERIMETER( <i>layer</i> ), or LENGTH( <i>layer</i> ) function in a DFM Analyze operation may not be a derived error layer.
DFM9	A DFM Analyze operation has an expression specified but no constraint specified.
DFM10	A DFM Analyze operation has a constraint specified but no expression specified.
DFM11	A DFM Transition operation is missing a SPACE1, SPACE2, SPACE3, ENC1, ENC2, or SIZE3 keyword. The name of the missing keyword is placed into the message.
DFM12	The STEP parameter in a DFM Transition operation may not be specified unless WINDOW is also specified.
DFM13	Both WINDOW and STEP are specified in a DFM Transition operation and a STEP value exceeds the corresponding WINDOW value or does not evenly divide it. The errant STEP value is placed into the message.
DFM14	A MAXIMUM, WINDOW, STEP, BY CELL, NOEMPTY, NOPSEUDO, INSIDE OF, INSIDE OF EXTENT, or INSIDE OF LAYER keyword is present in a DFM Transition operation, but RDB [ONLY] has not been specified. The name of the errant keyword is placed into the message.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM15	The OUTPUT1, OUTPUT2, or OUTPUT3 keyword has been specified in a DFM Transition operation with RDB ONLY also specified. The name of the errant keyword is placed into the message.
DFM16	A WINDOW, STEP, INSIDE OF, INSIDE OF EXTENT, or INSIDE OF LAYER keyword is present in a DFM Transition operation and RDB [ONLY] BY CELL has been specified. The name of the errant keyword is placed into the message.
DFM17	The NOPSEUDO keyword has been specified in a DFM Transition operation with an RDB [ONLY] specification without BY CELL. The name of the errant keyword is placed into the message.
DFM18	The STEP parameter in a DFM Measure operation may not be specified unless WINDOW is also specified.
DFM19	Both WINDOW and STEP are specified in a DFM Measure operation and a STEP value exceeds the corresponding WINDOW value or does not evenly divide it. The errant STEP value is placed into the message.
DFM20	A MAXIMUM, WINDOW, STEP, BY CELL, NOEMPTY, NOPSEUDO, INSIDE OF, INSIDE OF EXTENT, or INSIDE OF LAYER keyword is present in a DFM Measure operation but RDB [ONLY] has not been specified. The name of the errant keyword is placed into the message.
DFM21	A WINDOW, STEP, INSIDE OF, INSIDE OF EXTENT, or INSIDE OF LAYER keyword is present in a DFM Measure operation and RDB [ONLY] BY CELL has been specified. The name of the errant keyword is placed into the message.
DFM22	The NOPSEUDO keyword has been specified in a DFM Measure operation with an RDB [ONLY] specification without BY CELL. The name of the errant keyword is placed into the message.
DFM23	The DFM Measure operation has either space or width not specified.
DFM24	The S keyword is not specified in a DFM Critical Area Operation.
DFM25	The STEP keyword may not be specified without the WINDOW keyword in a DFM Critical Area operation.
DFM26	A DFM Critical Area STEP value must evenly divide the corresponding WINDOW value.
DFM27	A DFM Critical Area keyword is invalid without the RDB keyword. The name of the offending keyword appears in the message.
DFM28	A DFM Critical Area keyword is invalid with the RDB ONLY keyword. The name of the offending keyword appears in the message.
DFM29	A DFM Critical Area keyword is invalid with the BY CELL keyword. The name of the offending keyword appears in the message.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM30	A DFM Critical Area keyword is invalid with the BY NET keyword. The name of the offending keyword appears in the message.
DFM31	A DFM Critical Area keyword is invalid without the BY CELL keyword. The name of the offending keyword appears in the message.
DFM32	The SHORT or OPEN keywords may not be specified in a DFM Critical Area operation with more than one input layer.
DFM33	One of the following keywords must be specified in a DFM Critical Area operation: SHORT, OPENWIRE, OPENUWIRE, or OPENVIA.
DFM34	A required secondary keyword for the DFM Grow operation is missing.
DFM35	A DFM Analyze keyword is invalid with the BY CELL keyword. The name of the offending keyword appears in the message.
DFM36	A DFM Analyze keyword is invalid without the BY CELL keyword. The name of the offending keyword appears in the message.
DFM37	A DFM Critical Area keyword is invalid with the EXCLUDE EDGE keyword. The name of the offending keyword appears in the message.
DFM38	Cannot establish MASK connectivity of the output layer in the displayed DFM NODAL RuleCheck.
DFM39	The displayed token was found where a layer was expected.
DFM40	Invalid call of a per-geometry function in an argument expression to a per-geometry function.
DFM41	In an argument expression to a per-geometry function, invalid call of a built-in function.
DFM42	More than one layer was specified in an argument expression to SUM(), PROD(), or VECTOR().
DFM43	More than one of AREA/PERIMETER/LENGTH(layer) was specified in an argument expression to SUM() or PROD().
DFM44	An argument expression to SUM(), PROD(), MIN() or MAX() does not contain a geometry-specific term.
DFM45	A polygon-type layer parameter in an expression may not be used in EC, EW, ECMIN, ECMAX, or ANGLE functions.
DFM46	An edge-type layer parameter in an expression may not be used in EC, EW, or ANGLE functions.
DFM47	A property name is missing in a DFM Property operation.
DFM48	Incorrect input layer type in a DFM Property operation. All input layers must have the same type.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM49	Invalid function call in a DFM Property expression.
DFM50	Missing FILLSHAPE specification in a DFM Fill operation.
DFM51	Missing STEP specification in a DFM Fill operation.
DFM52	Layer spacing value must be non-negative in a DFM Fill operation.
DFM53	Missing layer in a SPACE specification in a DFM Fill operation.
DFM54	A FILLSHAPE must have at least two sets of coordinates in a DFM Fill operation.
DFM55	Odd number of coordinate values. The number of coordinate values must be even in a DFM Fill operation.
DFM56	Duplicate density expression in a DFM Fill operation.
DFM57	Either a density, magnitude, or gradient constraint is missing in a DFM Fill operation.
DFM58	All but the last FILLSHAPE must have SHAPESPACE specified in a DFM Fill operation.
DFM59	Invalid EFFORT value in a DFM Fill operation. The EFFOR value must be no greater than 10 and no less than 1.
DFM60	The displayed token was found where a function name was expected.
DFM61	The displayed token was found where a "(" was expected.
DFM62	The displayed token was found where a function argument name was expected.
DFM63	The displayed token was found where a ")" was expected.
DFM64	The displayed token was found where a "[" was expected.
DFM65	The displayed token was found where "Table" was expected.
DFM66	The displayed token was found where an x-coordinate was expected.
DFM67	The displayed token was found where a y-coordinate was expected.
DFM68	The displayed token was found where a "]" was expected.
DFM69	At least one argument is required for a DFM Function call.
DFM70	Duplicate function definition name.
DFM71	Table x-coordinates must be monotonically increasing.
DFM72	The displayed token was found where a "{" was expected.
DFM73	The displayed token was found where a "}" was expected.
DFM74	Table must have at least two x-y coordinate pairs.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM75	Attempt to re-define the displayed built-in function.
DFM76	An argument name cannot be the same as a built-in function.
DFM77	There is missing table data following the displayed token.
DFM78	Invalid interpolation style. Must be LINEAR or SPLINE.
DFM79	The displayed token was found where a function body was expected.
DFM80	Duplicate argument name in a DFM Function expression.
DFM81	The displayed keyword was used for a DFM Database file name.
DFM82	The displayed layer selector for a DFM Database operation was used outside of brackets ( “[ ] ” ).
DFM83	Unknown layer selector for a DFM Database operation.
DFM84	The displayed token was found where a formal layer parameter was expected.
DFM85	Invalid function call in a DFM Function expression.
DFM86	A reference to a non-layer formal parameter was found where an explicit formal layer parameter is required.
DFM87	No layer type is valid for all uses of a formal layer parameter.
DFM88	A polygon-type layer is invalid in a call to DFM Function.
DFM89	An edge-type layer is invalid in a call to DFM Function.
DFM90	An error-type layer is invalid in a call to DFM Function.
DFM91	Exactly one numeric parameter is required when a DFM FUNCTION body uses the displayed token.
DFM92	Recursion detected involving a call to the displayed DFM FUNCTION.
DFM93	The displayed token was found where a “?” was expected.
DFM94	The displayed token was found where a “:” was expected.
DFM95	The displayed token was found where an argument-type (e.g., “number” or “layer”) was expected.
DFM96	The displayed token was found where a “,” was expected.
DFM97	The displayed token was found where a string argument was expected.
DFM98	At least one numeric parameter is required when a DFM Function body uses the displayed token.
DFM99	A non-numeric parameter type is invalid when a DFM Function body uses the displayed token.
DFM100	The displayed token was found where an input coordinate was expected.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM101	The displayed token was found where a “::” was expected.
DFM102	The displayed token was found where an output coordinate was expected.
DFM103	A DFM Fill density expression must include _FILL_ layer.
DFM104	A WINDOW parameter is either missing or has zero size.
DFM105	Offset numeric parameter must be non-negative.
DFM106	The SHAPESPACE numeric parameter must be non-negative.
DFM107	The STEP numeric parameter must be non-negative.
DFM108	Invalid function call in a DFM Fill expression.
DFM109	The displayed layer already exists in the database.
DFM110	An expression in DFM Property with the INTERSECT or INSIDE OF options cannot call PROPERTY(<layer>,<propName>).
DFM111	Invalid function call in a DFM Analyze expression.
DFM112	ECMIN or ECMAX can only be called in argument to VECTOR()
DFM113	Invalid function call in a DFM Property operation.
DFM114	OVERLAP may not be specified in a DFM Analyze operation without NUMWIN.
DFM115	Cannot specify both WINDOW and NUMWIN in a DFM Analyze operation.
DFM137	FILLSHAPE is too big relative to the WINDOW.
DFM138	Malformed polygon specified.
DFM139	Maximum stretch size is smaller than fillshape dimension.
DFM150	No matching DFM Spec Fill found.
DFM151	Missing SPEC name.
DFM152	The displayed SPEC name is already defined.
DFM153	Magnitude constraint value is outside of the range defined by density and/or gradient constraints.
DFM154	Wrong number of coordinates specified for the displayed fillshape type.
DFM155	The stretch step is too small for the displayed fillshape.
DFM156	One of the max stretch values must be 0 for the displayed fillshape.
DFM157	Autorotate can only be used with a STRETCHFILL or RECTFILL type of fillshape.
DFM158	Output name is missing after the displayed token.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM159	Invalid parameter for the displayed statement or keyword.
DFM160	Missing parameter for the displayed statement or keyword.
DFM161	Invalid geometric data for the displayed statement or keyword.
DFM162	Nets cannot be specified if NODAL is not specified for the displayed operation or keyword.
DFM163	Nets must be specified if NODAL is specified for the displayed operation or keyword.
DFM175	Conflicting datatypes for the displayed variable.
DFM176	Found a non-numeric expression where only a numeric expression is valid.
DFM177	Found a non-string argument where only a string argument is valid.
DFM178	Found an invalid datatype for the displayed operator.
DFM179	The displayed argument must be an vector of numbers.
DFM180	Call to the displayed built-in function is valid only in a DFM Property expression.
DFM181	Conflicting datatypes for arguments to the ?: operator.
DFM182	Conflicting expression value types for a property.
DFM183	Cannot specify a constraint for a vector-valued property expression.
DFM184	Found non-vector expression when only a vector expression is valid.
DFM186	The displayed argument must be a number.
DFM187	The displayed keyword must be followed by a left bracket ( [ ).
DFM188	Illegal argument for ANNOTATE.
DFM189	Illegal option with the NULL filename.
DFM192	The specified OUTPUT is not defined in DFM Spec Fill.
DFM193	The analysis/gradient window size is not a multiple of the step.
DFM194	The gradient step is not a multiple of the window step.
DFM195	The step cannot be bigger than the window size.
DFM196	Too few arguments specified.
DFM201	At least one input layer must be specified for DFM Property.
DFM202	The displayed function can be called only in an argument expression to a per-shape function.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM213	The displayed DFM Critical Area keyword is invalid with REGION OCTAGONAL.
DFM214	The displayed DFM Critical Area keyword is invalid with METRIC SQUARE.
DFM215	INSIDE BY or OUTSIDE BY must be specified for DFM Expand Edge.
DFM218	The number of arguments to SELECT_MERGE_XXY must be even.
DFM219	The displayed argument must be a number.
DFM220	Found a non-NETID expression where only a NETID expression is valid.
DFM221	The displayed arguments must have the same datatype.
DFM222	The displayed argument must be a vector.
DFM223	Only the primary layer can be used when the displayed token is called outside a per-geometry function.
DFM224	Invalid secondary (non-primary) layer in a DFM Function call.
DFM225	Optional third argument to PROPERTY() cannot be used in an argument to a per-shape expression.
DFM226	An expression in DFM Property with the INTERSECT or INSIDE OF options cannot call NETPROPERTY or NETVPROPERTY.
DFM230	“SPATIAL SAMPLE spec_name” expected.
DFM231	CONVERGENCE constraint must be “<” or “<=”.
DFM232	Only the primary layer can be used if [,<ordinal>] is not specified.
DFM262	Unexpected number of output values.
DFM263	A size property must be specified for DFM Size.
DFM264	If either INSIDE OF or OUTSIDE OF are specified, there must be a second layer for DFM Size.
DFM265	Either INSIDE OF or OUTSIDE OF must be specified if there are two layers for DFM Size.
DFM266	CONVERGENCE constraint value cannot be greater than 1.0.
DFM267	The displayed function can be called only in an argument expression to a per-shape function in a DFM Property operation.
DFM268	The Precision specification statement database precision does not match the precision from the DFM database.
DFM269	INSIDE OF cannot be combined with CONNECTED, NOT CONNECTED, or BY NET.
DFM270	Expression in DFM Property with INSIDE OF option cannot call NETID().

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM271	Missing parameter for keyword.
DFM272	DFM Connectivity requires a mode keyword.
DFM273	The Variable specification statement does not match that from the DFM database.
DFM274	Cannot use both WINDOW and NUMWIN with DFM Spec Spatial Sample.
DFM275	The displayed DFM Critical Area keyword is invalid with SPATIAL SAMPLE.
DFM276	DFM Critical Area SPATIAL SAMPLE requires a SPEC name.
DFM277	Selector for DFM Database outside of brackets ("[ ]").
DFM278	Duplicate suffix.
DFM279	Missing file name after the displayed keyword.
DFM280	The displayed file can not be found or opened.
DFM282	The displayed token was found where "LENGTH" was expected.
DFM283	The displayed function can be called only in DFM Property.
DFM284	The displayed function uses [,ordinal] to fetch a property value. This is invalid when called by a DFM Analyze expression.
DFM285	Unsupported operation for the platform.
DFM290	The displayed token was found where a comma was expected.
DFM291	Tcl function could not be loaded
DFM292	Tcl interpreter could not be created.
DFM293	The displayed DFM Transform option requires one numeric argument.
DFM294	The displayed DFM Transform option requires two numeric arguments.
DFM295	Missing required option for DFM Transform.
DFM296	In a DFM Transform operation, CENTER was specified without a ROTATION specification.
DFM297	Invalid REPEAT value. Must be no greater than 10 and no less than 1.
DFM298	Either the size or step must be specified as a property name for DFM Size.
DFM299	Either INSIDE BY, OUTSIDE BY, or EXTEND BY must be specified as a property name for DFM Expand Edge.
DFM300	Invalid syntax found in the displayed function: missing comma.
DFM301	The displayed argument must be a vector of net IDs.
DFM302	BY must be followed by a property name for DFM Stamp.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM304	The displayed argument must be a vector of numbers with tuple-size 3.
DFM307	INSIDE BY or OUTSIDE BY must be specified for DFM Shift Edge.
DFM308	Either INSIDE BY, OUTSIDE BY, or EXTEND BY must be specified as a property name for DFM Shift Edge.
DFM309	For FILLSTACK, at least one metal layer must come before any VIA layers.
DFM310	For FILLSTACK, a shape name must be used. Coordinates are not allowed.
DFM311	Too many OUTPUT specifications for FILLSTACK (maximum of 24).
DFM312	Minimum number of metal layers for FILLSTACK is 2.
DFM313	Minimum number of via layers for FILLSTACK is 1.
DFM314	MAXSTACK must not be greater than the number of metal layers for FILLSTACK.
DFM315	MAXSTACK must be greater than or equal to MINSTACK.
DFM316	Top-level optimizer is not allowed with the displayed token.
DFM317	Missing or invalid TVF FUNCTION name for the DFM YS Autostart specification statement.
DFM318	Tcl procedure must follow the TVF function name for the DFM YS Autostart specification statement.
DFM319	The displayed argument must be a vector of numbers with tuple-size 2.
DFM320	Tcl interpreter could not be initialized.
DFM321	Tcl bytecode could not be loaded.
DFM322	Compilation of Tcl library was unsuccessful.
DFM323	A Tcl script cannot be called in the context of a DFM Fill statement.
DFM330	Property name missing from function call.
DFM331	NUMBER/STRING must follow PROPERTY keyword for the DFM Read operation.
DFM332	Only an original layer can be specified.
DFM333	A property name must be found after the displayed argument.
DFM334	A property name or a reference number must be found after the displayed keyword.
DFM335	DFM Read syntax implies a GDS layout which differs from the system layout.
DFM336	DFM Read syntax implies an OASIS layout which differs from the system layout.

**Table 7-8. DFM Operation Errors**

Error #	Explanation
DFM338	PSPROPERTY() third parameter ("ordinal") must be omitted when PSPROPERTY() is used in an argument to a per-shape function.
DFM339	SPROPERTY() third parameter ("ordinal") must be omitted when SPROPERTY() is used in an argument to a per-shape function.
DFM340	VPROPERTY() third parameter ("ordinal") must be omitted when VPROPERTY() is used in an argument to a per-shape function.
DFM341	NETPROPERTY() third parameter ("ordinal") must be omitted when NETPROPERTY() is used in an argument to a per-shape function.
DFM342	NETVPROPERTY() third parameter ("ordinal") must be omitted when NETVPROPERTY() is used in an argument to a per-shape function.
DFM344	SPLIT ALL is invalid when a property is referenced on the primary layer.
DFM345	SPLIT [ALL] is invalid when a property is referenced on any secondary layer of type 3 (edge cluster).
DFM347	DFM Read requires unique DFM property names.
DFM348	SPROPERTY() and PSPROPERTY() cannot be used with BY CELL or BY NET.
DFM349	NETID() second parameter ("ordinal") must be omitted when NETID() is used in an argument to a per-shape function.
DFM350	The displayed function uses [,<ordinal>] to fetch a netid value -- invalid when called by a DFM Analyze expression.
DFM353	Cannot reference a property starting with “-” or “+” (non-persistent DFM properties).
DFM354	Cannot reference the displayed property.
DFM355	Property references only allowed within the context of a DFM Property statement.
DFM356	Cannot reference the displayed property of different type.
DFM357	The displayed secondary keyword is missing for the DFM Expand Enclosure operation.
DFM358	The displayed keyword for DFM Expand Enclosure is superfluous.
DFM359	Either LINEENDMAX or SIDEMAX must be specified for the displayed keyword.
DFM360	The displayed Tcl function cannot be bound to more than one DFM Function.
DFM361	BACKUP may not be specified in DFM Analyze unless WINDOW is specified and NUMWIN is not used.

## Device Property Specification Errors

**Table 7-9. Device Property Specification Errors**

Error #	Explanation
DPR1	Only definitions for element names D, C, R, MN, MP, MD, and ME can have a numeric parameter set.
DPR2	The element names R, MN, MP, MD, and ME take only a single number as a numeric parameter set. For R, it is the resistivity and for the others, it is the effective width factor. No additional numbers are allowed.
DPR3	The numeric parameter set specification for element name C must contain either just the area capacitance factor, or the area capacitance factor followed by the perimeter capacitance factor. No additional numbers are allowed.
DPR4	The syntax element displayed was found where a valid assignment statement, IF statement, or compound statement beginning with a left brace ( { ) was expected.
DPR5	In a DEBUG statement, the first number and hyphen (-) of a debug range was found, but the second number was missing. A debug range must be either a single number or a pair of numbers separated by a hyphen.
DPR6	A number must appear immediately following the DEBUG keyword and immediately following each comma in the DEBUG statement.
DPR7	The syntax element shown appears where a closing brace was expected to terminate a compound statement.
DPR8	A left parenthesis must follow the keyword IF and all function name keywords.
DPR9	The syntax element displayed could not be parsed in the current context, but a right parenthesis would be valid at this point.
DPR10	The expressions used in an IF test must contain relational tests possibly combined with logical operators to form compound tests.
DPR12	Commas must be used to separate items in lists and the arguments of functions.
DPR13	With the exception of the DEBUG statement, the first statement in a property specification must be a PROPERTY statement specifying the properties to be computed.
DPR14	A property identifier must immediately follow the keyword PROPERTY and each comma in the PROPERTY statement.
DPR15	Each property identifier can appear only once in the list of the PROPERTY statement.
DPR16	In an assignment statement, the item just to the left of the assignment operator ( = ) must be a property name as declared in the PROPERTY statement, or an identifier representing a local variable.

**Table 7-9. Device Property Specification Errors (cont.)**

Error #	Explanation
DPR17	The syntax element shown is not the name of a pin or layer appearing in this Device operation, yet it appears as the argument of a function where a pin name or layer name is required.
DPR18	The property specification appears to end just prior to the syntax element shown, but no closing right bracket was found there.
DPR19	The syntax element shown was found where a constant, local variable, process variable, or numeric valued function was expected.
DPR20	The process variable referenced at this point must have a numeric value, but this one does not.
DPR21	The right-hand side of the statement contains the variable shown. However, this variable has either not been initialized in a prior statement, or has not been initialized in every IF/ELSE path leading to the current statement. For example, in the following, before reaching the statement A = X, the variable X would not be initialized if the IF condition were FALSE: <code>if(b == 0)     X = 1     A = X</code>
DPR22	The syntax element shown is not the name of a pin appearing in this Device operation, yet it appears as the argument of a function where a pin name is required.
DPR23	The variable displayed is used on both the left and right sides of the current statement. It must have been given a value before reaching the current statement so that the value can be used in the right-hand side of the current statement.
DPR24	The variable shown is assigned a value by the current statement. However, this variable has been identified as a process variable. Changing the value of a process variable is not allowed from within a property computation.
DPR25	The syntax element shown appeared as an argument to a function where a net name was expected but cannot be interpreted as a net name. It is best to surround the net name with quotation marks ("").
DPR35	All properties declared in the PROPERTY statement must be assigned a value in the property computation. The property name shown was never assigned a value.
DPR36	All properties declared in the PROPERTY statement must be assigned a value in the property computation. The property name shown was assigned a value in some of the IF/ELSE cases but not in all. That is, it is possible to find a path through the program that never assigns a value to the variable. You must be sure it receives a value in all cases.
DPR37	The function shown requires two arguments, but they cannot be identical.

**Table 7-9. Device Property Specification Errors (cont.)**

Error #	Explanation
DPR38	The function shown requires two arguments, but they cannot represent shapes on the same layer. For example, using different pin names from the same layer is not allowed, nor is using a pin name from a layer together with the layer itself. In all these disallowed cases, the function would return a trivial value that can be expressed in another way. For example, if A and B are pins on layer L, then PERIMETER_COINCIDE(A, B) would always be 0 (zero) because A and B would have to be disjoint, and PERIMETER_COINCIDE(A, L) would be the same as PERIMETER(A) since pin A lies on layer L.
DPR39	The function shown requires two pin or layer arguments and they cannot be identical.
DPR40	The function shown has pin or layer arguments and they cannot be associated with the same layer.
DPR41	A positive constant is missing from a place where one is required.
DPR42	A variable exceeds the limits for table size.
DPR43	A process variable is used in both vector and numeric constants. This is not allowed.
DPR44	A single vector variable has more than one vector assignment to it.
DPR45	The user-defined program causes an internal compiler error.
DPR46	The user-defined program causes an internal compiler error.
DPR47	A scope token (::) is missing where one is required.
DPR48	A vector value expression is missing where one is required.
DPR49	A string variable, string constant, or a string function is missing where one is required.
DPR50	A function requires a vector variable.
DPR51	A vector expression has two vector variables. This is not allowed.
DPR52	A typed variable is missing where one is required.
DPR53	The layer is disallowed in the context in which it appears.
DPR54	A string constant contains an embedded double quotation mark ("").
DPR55	The function shown is not allowed within a vector expression.
DPR56	A Device TEXT PROPERTY LAYER name is missing where one is required.
DPR57	A string constant is missing where one is required.
DPR58	A process variable resolves to a list of strings when it is required to be a single string.
DPR59	A layer argument is used where a pin name is needed.

**Table 7-9. Device Property Specification Errors (cont.)**

Error #	Explanation
DPR60	A property is used that cannot appear outside of a Device TVF function.
DPR61	A pin derived from a Device seed layer is used for a DFM Property calculation.
DPR62	The user-defined program has too many variables or temporary assignments. Fewer are needed or Device TVF should be used.
DPR63	A function identifier is missing where one is required.
DPR64	A string constant or variable was expected at this location.
DPR65	Invalid property name.
DPR66	A layer name was expected at this location.
DPR67	An assignment of incompatible data types occurs at this location.

## LVS Device Type Statement Errors

**Table 7-10. LVS Device Type Statement Errors**

Error #	Explanation
DTP1	An LVS Device Type statement is missing a pin mapping list.
DTP2	An LVS Device Type statement has superfluous pin mapping lists.
DTP3	An LVS Device Type statement has superfluous LAYOUT or SOURCE keywords.
DTP4	An LVS Device Type statement has a duplicate pin name.
DTP5	There are duplicate LVS Device Type statements with the same device name.
DTP6	An LVS Device Type statement has an invalid pin mapping list.
DTP7	An LVS Device Type statement has an invalid pin name.
DTP8	An LVS Device Type statement references a Device type, but the corresponding Device definition is missing a required pin as specified in the LVS Device Type statement.
DTP9	An LVS Device Type statement references an undefined Device type. A Device statement is missing that corresponds to the Device type named in the LVS Device Type statement.

## Device Layer Operation Errors

**Table 7-11. Device Layer Operation Errors**

Error #	Explanation
DVL1	A Device Layer operation is missing its element name parameter or has an invalid element name parameter.
DVL2	A model name within a Device Layer operation is not of the form “(” <name> “)”. The device element name appears in the message.
DVL3	A Device Layer operation has no corresponding Device definition in the rule file. The device element name appears in the message.
DVL4	More than one Device definition matches a Device Layer operation with the PROPERTY keyword. The device element name appears in the message.
DVL5	A pin is missing or is inconsistent for instances this device.
DVL6	The specified layer is not used for any devices.

## Environment Variable Errors

**Table 7-12. Environment Variable Errors**

Error #	Explanation
ENV1	<ul style="list-style-type: none"> <li>The string \$xx...xx in an Include statement pathname cannot be resolved as a non-null value in the shell environment. The erroneous string appears in the message.</li> <li>A <i>name</i> in a Variable specification statement with ENVIRONMENT specified cannot be resolved as a non-null value in the shell environment. The erroneous <i>name</i> appears in the message.</li> <li>A <i>name</i> in the Precision specification statement cannot be resolved as a non-null value in the shell environment. The erroneous string appears in the message.</li> <li>A <i>name</i> in the Layout Property Audit specification statement begins with “\$” but the remainder cannot be resolved as a non-null value in the shell environment. The erroneous string appears in the message.</li> </ul>
ENV2	A <i>name</i> in a Variable specification statement with ENVIRONMENT specified is resolvable in the shell environment but cannot be converted to a numeric value. The erroneous <i>name</i> appears in the message.

## ERC Errors

**Table 7-13. ERC Errors**

Error #	Explanation
ERC1	The EXCLUDE SUPPLY keyword of the Pathchk and ERC Pathchk statements requires that POWER or GROUND be specified also.
ERC2	Invalid object specified with the operation.

## Connectivity Extraction Errors

**Table 7-14. Connectivity Extraction Errors**

Error #	Explanation
EXT1	More than one Label Order operation for a given connectivity type was encountered in the rule file.
EXT2	A layer cannot have its connectivity verified in the Direct connectivity set according to the rules presented under the appropriate operation. The erroneous layer name or number appears in the error message.
EXT3	A layer cannot have its connectivity verified in the mask connectivity set according to the rules presented under the appropriate operation. The erroneous layer name or number appears in the error message.

**Table 7-14. Connectivity Extraction Errors (cont.)**

Error #	Explanation
EXT4	The derivation tree for a layer within a Connect or Sconnect operation includes a Net, Not Net, Net Area, Net Area Ratio, Stamp, constrained polygon topological operation with the BY NET keyword, [Not] With Neighbor operation with the CONNECTED or NOT CONNECTED keyword, Net Interact operation or nodal dimensional check operation. The name of the erroneous layer appears in the error message. This error is waived in Calibre nmDRC/nmDRC-H when DRC Incremental Connect YES is specified.
EXT5	A layer within a connectivity extraction operation is also a member of a layer set within a connectivity extraction operation in the same connectivity set. The name or number of the erroneous layer appears in the error message.
EXT6	A layer set within a connectivity extraction operation has a non-empty intersection with another non-identical layer set within a connectivity extraction operation in the same connectivity set. The name of the erroneous layer set appears in the error message.
EXT7	A Connect or Sconnect operation contains more than 32 input layers.
EXT8	DRC Incremental Connect YES was specified and either a layer parameter to a Connect operation has a forward reference in its definition, or a layer parameter to any layer operation has a forward reference in its definition to a different connectivity zone. The name of the erroneous layer appears in the error message.
EXT9	DRC Incremental Connect YES was specified and a Mask mode Sconnect operation appears prior to the last connectivity zone.
EXT10	An upper-layer parameter to a Sconnect operation is not in a Connect operation in the same connectivity set and does not appear as a lower layer in a Sconnect operation in the same connectivity set. The name of the erroneous layer appears in the error message.
EXT11	The lower-layer parameter to a Sconnect operation must not be in a Connect operation or be the contact layer in an sconnect operation in the same connectivity set. The name of the erroneous layer appears in the error message.
EXT12	A mask mode Label Order operation was specified when DRC Incremental Connect YES was specified.
EXT13	Invalid Sconnect chain. The layers specified in Sconnect statements lead to a conflict in uni-directional connectivity.
EXT14	The indicated layer must appear either in a Connect or Sconnect operation.

## LVS Filter Statement Errors

**Table 7-15. LVS Filter Statement Errors**

Error #	Explanation
FIL1	A LVS Filter statement lacks an element name parameter or has an invalid element name parameter.
FIL2	A model name contained in a LVS Filter statement is not in the form '(<name>)'. The element name of the erroneous LVS Filter statement appears in the error message.
FIL3	A SPICE value contained in a LVS Filter statement is not in the form '(<name>)' or the <name> is invalid. The property name associated with the SPICE value appears in the error message.
FIL4	The filter constraint is improperly specified, including mixed string and numeric types in interval constraints and interval constraints where A >= B. The element name of the erroneous LVS Filter statement appears in the error message.
FIL5	A LVS Filter statement with a property SPICE value has a string-valued filter constraint. The element name of the erroneous LVS Filter statement appears in the error message.
FIL6	A LVS Filter statement lacks either a SHORT or an OPEN specification. The element name of the erroneous LVS Filter statement appears in the error message.
FIL7	A Mask mode LVS Filter statement has a SPICE value specified. The property name associated with the SPICE value appears in the error message.
FIL8	A Mask mode LVS Filter statement has no corresponding device definition in the rule file. The element name of the erroneous LVS Filter statement appears in the error message.
FIL9	A property in a Mask mode LVS Filter statement is not computed in all corresponding rule file device definitions. The property name appears in the error message.
FIL10	A property in a Mask mode LVS Filter statement is computed in some corresponding rule file device definitions with a type different from the type (string or numeric) specified in the statement. The property name appears in the error message.
FIL11	An LVS Filter statement specifies a SHORT pin that is inconsistent with other LVS Filter statements.
FIL12	An LVS Filter statement specifies a SHORT pin is swappable with other pins that are not shorted to it.
FIL13	A shorted pin is not present in the Device definition.

## Rule Check Group Errors

**Table 7-16. Rule Check Group Errors**

Error #	Explanation
GRP1	A rule check group specification lacks a name parameter or has an invalid name parameter.
GRP2	A rule check group specification has no definition or its definition contains no rule check statements, even when fully flattened. The erroneous rule check group name appears in the error message.
GRP3	The name of a rule check group duplicates that of another rule check group in the rule file. The erroneous rule check group name appears in the error message.
GRP4	An operand within a rule check group specification is a keyword, a number, or is not the name of any rule check statement or rule check group in the rule file. The erroneous entity appears in the error message.
GRP5	The name of a rule check group is the same as that of a rule check statement. The erroneous name appears in the message.

## Included Rule File Errors

**Table 7-17. Included Rule File Errors**

Error #	Explanation
INCL1	An Include file cannot be opened for read access. The name of the problem file appears in the error message.
INCL2	A file has been included recursively. The name of the erroneous file appears in the error message.
INCL3	An Include keyword was not followed by an operand on the same line.
INCL4	An Include keyword was followed by more than one operand on the same line.

## Operation Input Errors

**Table 7-18. Operation Input Errors**

Error #	Explanation
INP1	A superfluous or invalid input object for a specification statement or operation was encountered. The erroneous input object appears in the error message.
INP2	An operation has less than the required number of input layers. The name of the erroneous operation appears in the error message.
INP3	The Magnify, Shift, Expand Text, Size, Rotate, or Stamp operation is missing a BY keyword. The name of the erroneous operation appears in the error message.

**Table 7-18. Operation Input Errors (cont.)**

Error #	Explanation
INP4	<p>The following are missing name parameters:</p> <ol style="list-style-type: none"> <li>1. (Not) Net operation is missing its net name parameter.</li> <li>2. (Not) With Text or Expand Text operation is missing its text name parameter.</li> <li>3. Extent Cell or (Not) Inside Cell operation is missing its cell name parameter.</li> <li>4. Net Area Ratio Print operation is missing its filename parameter.</li> <li>5. PRINT keyword in a Density operation.</li> <li>6. RDB keyword in the Net Area Ratio, Density, or DFM operation is missing its filename parameter.</li> <li>7. AREF keyword in the DRC Check Map specification statement is missing its cell name parameter.</li> <li>8. Filename parameter in a Fracture or MDP operation.</li> </ol> <p>The name of the erroneous operation appears in the error message.</p>
INP5	<p>Any operation with two or more input layers has two identical input layers. There is no benefit in doing this and it is trapped as an unintentional error. The name of the erroneous operation appears in the error message.</p>
INP6	<p>A dimensional check, TDDRC, or DFM Measure operation has a superfluous edge-directed output specification [ ] or ( ). The name of the erroneous operation appears in the message.</p>
INP7	<p>A sequence of words forming a prefix of a multiple-word keyword (but not itself a keyword) is specified without the remainder of the keyword, or the keyword specification is incomplete. The erroneous word sequence appears in the error message.</p>
INP8	<p>An edge-directed output specification within a dimensional check, TDDRC, or DFM Measure operation is improperly formed, that is, not of the form [&lt;layer&gt;] or (&lt;layer&gt;).</p>
INP9	<p>A dimensional check, TDDRC, or DFM Measure operation contains both an edge-directed output specification ([&lt;layer&gt;] or (&lt;layer&gt;)) and a REGION or REGION EXTENTS keyword. The name of the erroneous operation appears in the error message.</p>
INP10	<p>The keyword must be followed by a subsequent layer parameter. For example, the BY keyword in a Stamp operation is not immediately followed by an input layer. The name of the errant keyword is placed into the message.</p>
INP11	<p>An operation or keyword requires one or more subsequent numeric parameters but has none, or an insufficient number of numeric parameters. For example, the BY keyword in a Size operation is not immediately followed by a numeric operand. Or, for example, the Rectangles operation does not have at least three numeric operands. The name of the errant keyword or the operation is placed into the message.</p>

**Table 7-18. Operation Input Errors (cont.)**

Error #	Explanation
INP12	The keyword requires a subsequent constraint parameter but has none. For example, the BY keyword in a Rectangle operation is not immediately followed by a constraint. The name of the offending keyword is placed into the message.
INP13	An Expand Edge operation has no expansion specification, that is, one of the secondary keywords: INSIDE BY, OUTSIDE BY, INSIDE BY FACTOR, OUTSIDE BY FACTOR, BY, or BY FACTOR.
INP14	An input layer to any operation, except Net Area, Net Area Ratio, or Ornet has an Ornet operation in its derivation path. The name of the erroneous layer appears in the error message.
INP15	A name does not follow the listed keywords: 1. PRINT [ONLY] keyword in a Density operation. 2. MAP keyword in a LITHO operation. 3. LINK keyword in the Sconnect operation. The name of the erroneous keyword appears in the error message.
INP16	The four coordinates X1, Y1, X2, Y2 following the INSIDE OF keyword in a Density, DFM, Rectangles, Fracture, or MDP operation do not represent a valid non-trivial orthogonal rectangle, that is, $X1 < X2$ and $Y1 < Y2$ . The name of the offending keyword is placed into the message.
INP17	Missing, incomplete, over-specified, or invalid endpoint specification for the CONVEX EDGE operation because of: 1. No endpoint specification. 2. Both simple and detailed endpoint specification. 3. ANGLE1(2) without ANGLE2(1). 4. LENGTH1(2) without ANGLE1(2).
INP18	Reserved.
INP19	Reserved.
INP20	The Grow or Shrink operation lacks any of the RIGHT BY, LEFT BY, TOP BY, or BOTTOM BY parameters.
INP21	The Rectangle Enclosure operation lacks specification of any GOOD or BAD parameters.
INP22	The (Not) With Neighbor operation has no spacing parameter.
INP23	The TDDRC operation has neither SPACE nor WIDTH specified.
INP24	The keywords ORTHOGONAL ONLY and OCTAGONAL ONLY may not be specified in the (Not) With Neighbor operation unless CENTERS is also specified.
INP25	Reserved.
INP26	Reserved.

**Table 7-18. Operation Input Errors (cont.)**

Error #	Explanation
INP27	Reserved.
INP28	OPPOSITE1, OPPOSITE2, OPPOSITE EXTENDED1, or OPPOSITE EXTENDED2 is specified in a one-input-layer dimensional check operation.
INP29	Parameters greater than 2 are not supported for COUNT.
INP30	EXCLUDE SHIELDED...COUNT 1 or 2 is specified with a shielding degree of 0, 1, or 2. The shielding degree must be 3 or greater.
INP31	EXCLUDE SHIELDED...COUNT 1 or 2 is only supported with the OPPOSITE metric.
INP32	Edge-directed output from the Offgrid operation is not supported with a polygon-type input layer plus the CENTERS keyword.
INP33	Polygon-directed output from the Offgrid operation is not supported with an edge-type input layer.
INP34	An OFFSET parameter in an Offgrid operation is incorrectly formed, that is, is not of the form OFFSET <value> <value> [INSIDE BY <value>   OUTSIDE BY <value>] or of the form OFFSET INSIDE BY <value>, or of the form OFFSET OUTSIDE BY <value>.
INP35	An OFFSET INSIDE BY or OFFSET OUTSIDE BY keyword appears in an Offgrid operation with polygon-type input. This is not supported.
INP36	A layer must be specified for this specification statement.

## Secondary Keyword Errors

**Table 7-19. Secondary Keyword Errors**

Error #	Explanation
KEY1	A secondary keyword was encountered twice in the same operation or statement, multiple specification of INSIDE OF, INSIDE OF EXTENT, WINDOW, or STEP keywords. The erroneous secondary keyword appears in the error message.
KEY2	A conflicting secondary keyword was specified in a statement or operation. For example, both PARALLEL ONLY and PERPENDICULAR ALSO are specified in a dimensional check operation. Or, for example, both INSIDE OF and INSIDE OF LAYER are specified in a Density operation. The name of the errant keyword is placed into the message.
KEY3	An OVERLAP or SINGULAR keyword appears in a dimensional check operation where at least one input layer is a derived edge layer. The name of the erroneous keyword appears in the error message.
KEY4	A MEASURE ALL or OVERLAP keyword appears in a one-layer dimensional check operation. The name of the erroneous keyword appears in the error message.
KEY5	The NOTCH or SPACE keyword appears in a two-layer dimensional check operation or a one-layer Internal operation. The name of the erroneous keyword appears in the error message.
KEY6	The CONNECTED or NOT CONNECTED keyword appears in a one-layer Internal operation. The name of the erroneous keyword appears in the error message.
KEY7	The CONNECTED or NOT CONNECTED keyword appears in a one-layer External operation with the NOTCH filter specified. The name of the erroneous keyword appears in the error message.
KEY8	An orientation, projection, or corner filter appears in a dimensional check operation that contains the INTERSECTING ONLY keyword. The name of the erroneous filter appears in the error message.
KEY9	An INTERSECTING ONLY keyword appears in a dimensional check operation that does not contain any specification for measurement of intersecting edges (an ABUT, OVERLAP, or SINGULAR keyword).
KEY10	The INSIDE ALSO keyword appears in a one-layer dimensional check operation, or a two-layer Internal operation. The name of the erroneous keyword appears in the message.
KEY11	The OUTSIDE ALSO keyword appears in a one-layer dimensional check operation, or a two-layer Internal or External operation. The name of the erroneous keyword appears in the message.

**Table 7-19. Secondary Keyword Errors (cont.)**

Error #	Explanation
KEY12	The MEASURE COINCIDENT keyword appears in a one-layer dimensional check operation, or a two-layer External operation. The name of the erroneous keyword appears in the message.
KEY13	An INSIDE ALSO keyword appears in an Enclosure operation where the first input layer is a derived-edge layer. The name of the erroneous keyword appears in the message.
KEY14	A bad constraint type is used with the STEP keyword.
KEY15	The STEP keyword may only be used for rule check output.
KEY16	The GOLDEN or RULE keywords may only be specified with WITH MATCH.

## LVS Annotate Devices Errors

**Table 7-20. LVS Annotate Devices Errors**

Error #	Explanation
LAD1	The Device element name is invalid.
LAD2	The Device model name is invalid.

## Layer Definition Errors

**Table 7-21. Layer Definition Errors**

Error #	Explanation
LAY1	A layer definition is defined in terms of itself (at some level). The erroneous layer definition name appears in the error message.
LAY2	The operation assigned in a layer definition is a connectivity extraction, parasitic extraction, or device recognition operation. The erroneous layer definition name appears in the error message.
LAY3	A layer definition does not have an assigned operation. The erroneous layer definition name appears in the error message.
LAY4	A layer definition name duplicates a name defined or encountered in an original layer specification statement. The erroneous layer definition name appears in the error message.
LAY5	A layer definition name duplicates that of another layer definition in the same scope. The erroneous layer definition name appears in the error message.
LAY6	An implicit layer definition contains one or more of the following: 1. Empty parenthesis. 2. {,}, or = inside the parenthesis. 3. A specification statement primary keyword inside the parenthesis. 4. A primary keyword (for example, Connect) representing a non-layer operation inside the parenthesis. 5. Multiple primary keywords at the same level inside the parenthesis. 6. No primary keyword inside the parenthesis, but a nested implicit layer definition.  The problem object appears in the error message.
LAY7	An implicit layer definition is not being used as the input layer to an operation.

## Layout Cell List Statement Errors

**Table 7-22. Layout Cell List Statement Errors**

Error #	Explanation
LCL1	A Layout Cell List statement has an invalid name.
LCL2	There are duplicate Layout Cell List names.
LCL3	A specification statement references a Layout Cell List name that does not exist.
LCL4	A Layout Cell List statement has an invalid argument specified. It expects a list name, followed by a pattern to match.
LCL5	There exist Layout Cell List statements that cause a circular reference to the same cell names.
LCL6	A Layout Cell List statement begins with an exclusive pattern match. It must begin with an inclusive pattern match.

## Layout Cell Match Rule Errors

**Table 7-23. Layout Cell Match Rule Statement Errors**

Error #	Explanation
LCMR1	A Layout Cell Match Rule <i>rule_name</i> parameter has been used more than once.
LCMR2	A Layout Cell Match Rule statement is missing a BY LAYER keyword specification.
LCMR3	Internal use.
LCMR4	A WITH MATCH RULE <i>rule_name</i> keyword set is specified for which there is no corresponding Layout Cell Match Rule <i>rule_name</i> .
LCMR5	MAXIMIM RESULTS is used and there is a missing or invalid parameter.

## Layout MOS Swappable Properties Errors

**Table 7-24. Layout MOS Swappable Properties Statement Errors**

Error #	Explanation
LMSP1	The same pair of s/d pin properties is specified more than once in the rule file for this statement.
LMSP2	The statement must specify exactly two property names.

## Location Errors

**Table 7-25. Location Errors**

Error #	Explanation
LOC1	A rule check comment was encountered outside of the brackets delimiting the body of a rule check statement.
LOC2	A layer operation which is not assigned as part of a layer definition was encountered outside of the brackets delimiting the body of a rule check statement. The primary keyword of the erroneous operation appears in the error message.
LOC3	A specification statement, connectivity extraction operation, parasitic extraction operation, or device recognition operation was encountered within the brackets delimiting the body of a rule check statement. The identifying keyword of the erroneous statement or operation appears in the error message.
LOC4	The statement or operation occurs outside the scope of a rule check block.

## LVS Push Devices Statement Errors

**Table 7-26. LVS Push Devices Statement Errors**

Error #	Explanation
LPDKP1	An LVS Push Devices statement with the KEEP PROPERTIES keyword has a missing or invalid component type.
LPDKP2	An LVS Push Devices statement with the KEEP PROPERTIES keyword has an improperly specified component subtype.
LPDKP3	An LVS Push Devices statement with the KEEP PROPERTIES keyword is lacking a list of properties.
LPDKP4	There are duplicate LVS Push Devices statements with the KEEP PROPERTIES keyword.
LPDSP1	An array may not be used as a Device property.

## LVS Property Initialize Statement Errors

**Table 7-27. LVS Property Initialize Statement Errors**

Error #	Explanation
LPI1	An LVS Property Initialize statement has a missing or invalid component type.
LPI2	An LVS Property Initialize statement has a missing user-specified program section.
LPI3	There are duplicate LVS Property Initialize statements.

**Table 7-27. LVS Property Initialize Statement Errors (cont.)**

Error #	Explanation
LPI4	An LVS Property Initialize statement has an incomplete user-specified program section.
LPI5	An LVS Property Initialize statement has duplicate PROPERTY statements in the user-specified program.
LPI6	An LVS Property Initialize statement has a misplaced ] character for the user-specified program.
LPI7	An LVS Property Initialize statement has an improperly specified component subtype.

## PERC Load Statement Errors

**Table 7-28. PERC Load Statement Errors**

Error #	Explanation
LPL1	A PERC Load statement has a missing or invalid TVF Function block name.
LPL2	A PERC Load statement has a missing or invalid INIT procedure name.
LPL3	A PERC Load statement has a missing or invalid rule check name.
LPL4	A PERC Load statement is specified more than once.

## LVS Property Map Statement Errors

**Table 7-29. LVS Property Map Statement Errors**

Error #	Explanation
LPM1	An LVS Property Map specification statement does not have an element name parameter or has an invalid element name parameter.
LPM2	An LVS Property Map specification statement does not have an input property specified or the specified input property is not a name. The element name of the erroneous LVS Property Map specification statement appears in the message.
LPM3	An LVS Property Map specification statement does not have a target property specified or the specified target property is not a name. The element name of the erroneous LVS Property Map specification statement appears in the message.
LPM4	A SPICE value within an LVS Property Map specification statement is not of the form “( <name> )”, or the <name> is not w, W, l, L, r, R, c, C, a, A, p, or P. The property name associated with the SPICE value appears in the message.
LPM5	Either LAYOUT or SOURCE must be specified in an LVS Property Map specification statement. The element name of the erroneous LVS Property Map specification statement appears in the message.
LPM6	An LVS Property Map specification statement shares the same element name, input database type, and property specification with another LVS Property Map specification statement in the same rule file. The element name of the erroneous LVS Property Map specification statement appears in the message.
LPM7	An LVS Property Map specification statement shares the same element name, input database type, and target property specification with another LVS Property Map specification statement in the same rule file, but the input property specifications differ. The element name of the erroneous LVS Property Map specification statement appears in the message.

**Table 7-29. LVS Property Map Statement Errors (cont.)**

Error #	Explanation
LPM8	An LVS Property Map specification statement shares the same element name, input database type, and input property specification with another LVS Property Map specification statement in the same rule file, but the target property specifications differ. The element name of the erroneous LVS Property Map specification statement appears in the message.
LPM9	An LVS Property Map specification statement has no corresponding device definition in the rule file. This error results from consistency checking against rule file device definitions. The element name of the erroneous LVS Property Map specification statement appears in the message.
LPM10	An input property in an LVS Property Map specification statement is not computed in all corresponding rule file device definitions. This error results from consistency checking against rule file device definitions. The erroneous property name appears in the message.
LPM11	An LVS Property Map specification statement property has a SPICE value specified. This error results from consistency checking against rule file device definitions. The name of the erroneous property appears in the message.
LPM12	An LVS Property Map property is not computed with the correct form in all corresponding Device definitions. Properties must be string or numeric, and they must be consistent in all corresponding Device definitions.

## PERC Property Statement Errors

**Table 7-30. PERC Property Statement Errors**

Error #	Explanation
LPP1	A PERC Property statement has a missing or invalid device element name.
LPP2	A PERC Property statement has an improperly specified device model name.
LPP3	A PERC Property statement has a missing or invalid property name.
LPP4	A PERC Property statement is specified more than once.
LPP6	A PERC Property statement specifies an undefined DEvice element.
LPP7	A PERC Property statement specifies an property that is not computed for all corresponding DEvice definitions.

## LVS Reduce Statement Errors

**Table 7-31. LVS Reduce Statement Errors**

Error #	Explanation
LRD1	An LVS Reduce statement is missing a property list after the TOLERANCE keyword in the user-defined program.
LRD2	An LVS Reduce statement has an invalid or incomplete TOLERANCE property name and tolerance number pair.
LRD3	An LVS Reduce statement has an invalid or improperly specified SPICE value. This error is deprecated.
LRD4	An LVS Reduce statement has a misplaced EFFECTIVE property name in the user-defined program.
LRD5	An LVS Reduce statement has a misplaced numeric scalar expression in the user-defined program.
LRD6	An LVS Reduce statement has a misplaced EFFECTIVE property name or simple constant in the user-defined program.
LRD7	An LVS Reduce statement has a misplaced ] or TOLERANCE keyword in the user-defined program.
LRD8	An LVS Reduce statement has a missing or invalid component type.
LRD9	An LVS Reduce statement has an improperly specified component subtype.
LRD10	An LVS Reduce statement has a misplaced PARALLEL or SERIES keyword.
LRD11	An LVS Reduce statement has a missing or invalid SERIES pin names near the indicated string.

**Table 7-31. LVS Reduce Statement Errors (cont.)**

Error #	Explanation
LRD12	An LVS Reduce statement with the NO keyword has a user-defined program specified with it.
LRD13	There are duplicate LVS Reduce statements.
LRD14	An LVS Reduce statement conflicts with another LVS Reduce Series or LVS Reduce Parallel statement, which applies to built-in device types.
LRD15	An LVS Reduce statement specifies a device component that has no corresponding Device definition or LVS Property Initialize statement.
LRD16	An LVS Reduce statement has duplicate series pin names.
LRD17	An LVS Reduce statement has an invalid built-in pin name for series devices.
LRD18	An LVS Reduce statement matches a Device definition component, but the Device definition lacks a pin name that is specified in the LVS Reduce statement.
LRD19	An LVS Reduce statement specifies series reduction for a built-in device type that does not support built-in reduction.
LRD20	An LVS Reduce statement specifies series reduction of pins that are swappable with non-series pins.
LRD21	An LVS Reduce statement specifies a property that is not present in all corresponding Device definitions or LVS Property Initialize statements.
LRD22	An LVS Reduce statement specifies a property that is not computed with the correct type in all corresponding Device definitions or LVS Property Initialize statements.
LRD23	An LVS Reduce statement has an invalid TOLERANCE STRING property name.
LRD24	An LVS Reduce statement has a missing TOLERANCE STRING property list.
LRD25	An LVS Reduce statement has a missing TOLERANCE STRING property list.
LRD26	An LVS Reduce statement has a duplicate EFFECTIVE statement in the user-defined program.
LRD27	An LVS Reduce statement has a duplicate TOLERANCE or TOLERANCE STRING property name.
LRD28	Reserved.
LRD29	Reserved.
LRD30	An LVS Reduce statement has a misplaced EFFECTIVE statement in the user-defined program.

## LVS Short Equivalent Nodes Statement Errors

**Table 7-32. LVS Short Equivalent Nodes Errors**

Error #	Explanation
LSEN1	An LVS Short Equivalent Nodes SPLIT statement and an LVS Reduce Split Gates YES statement are in the rule file. This is not allowed.

## Litho Operation Errors

**Table 7-33. Litho Operation Errors**

Error #	Explanation
LTH1	The Litho operation is missing a required secondary keyword.
LTH2	The number of input layers specified for the Litho operation with the PSMGATE keyword was other than two or three.
LTH3	The keyword FILE cannot be specified in a Litho operation when the keyword PSMGATE is specified.
LTH4	The keyword FILE must be specified in a Litho operation when the keyword ORC, OPC, OPCVERIFY, PSMGATE, or PRINTIMAGE is specified.
LTH5	There must be exactly two or three input layers for a Litho operation with PSMGATE specified.
LTH6	The FILE parameter in a Litho operation is not followed by a filename or an inlined file.
LTH7	The Fracture operation is missing a required secondary keyword.
LTH8	The FILE parameter is missing for Fracture or MDP operation.
LTH9	The FILE parameter in a Fracture or MDP operation is not followed by an inlined file.
LTH10	The FILE parameter for a LITHO operation has a syntax or semantic error.
LTH11	The FILE parameter for a Fracture operation has a syntax or semantic error.
LTH12	The FILE parameter for an MDP operation has a syntax or semantic error.
LTH13	Either SMALLSIDEDOUTTRAP or SPLITCD must be specified in an MDPstat operation.
LTH14	No keyword has been specified in an MDP operation.
LTH15	The file parameter for a Density Convolve operation has a syntax or semantic error.
LTH16	A litho operation has API specified without OPC, ORC, or PRINTIMAGE.
LTH17	The DBCLASSIFY operation requires a FILE parameter.

**Table 7-33. Litho Operation Errors (cont.)**

Error #	Explanation
LTH18	The DBCLASSIFY FILE parameter must be followed by a filename or a left bracket.
LTH19	There is a problem with the DBCLASSIFY FILE.

## Macro Errors

**Table 7-34. Macro Errors**

Error #	Explanation
MAC1	A DMACRO keyword is not followed by a name.
MAC2	A DMACRO definition is missing a left brace ( { ). The errant DMACRO name is listed in the message.
MAC3	A DMACRO name is duplicated in the rule file. The errant DMACRO name is listed in the message.
MAC4	A DMACRO name is invalid (that is, it's a keyword, numeric constant, or other invalid construct). The errant name is listed in the message.
MAC5	A DMACRO argument is duplicated in a DMACRO definition. The errant argument is listed in the message.
MAC6	A CMACRO keyword is not followed by a name.
MAC7	A CMACRO argument is invalid (that is, it's a keyword or other invalid construct). A valid DMACRO name or a numeric constant is required. The errant argument is listed in the message.
MAC8	A CMACRO name does not match any DMACRO name in the rule file. The errant name is listed in the message.
MAC9	A CMACRO has fewer arguments than its corresponding DMACRO. The errant name is listed in the message.
MAC10	A DMACRO definition is called recursively. The errant name is listed in the message.

## LVS Map Device Statement Errors

**Table 7-35. LVS Map Device Statement Errors**

Error #	Explanation
MDV1	An LVS Map Device statement is missing an original Device component type.
MDV2	An LVS Map Device statement has an invalid original Device component type. The component type must be a valid name (not a layer, number, SVRF keyword, or a string containing reserved characters, unless quoted).
MDV3	An LVS Map Device statement requires an <i>old_subtype</i> parameter.
MDV4	An LVS Map Device statement has an invalid <i>old_subtype</i> parameter.
MDV5	An LVS Map Device statement has an improperly specified <i>old_subtype</i> parameter.
MDV6	An LVS Map Device statement is missing an new Device component type.

**Table 7-35. LVS Map Device Statement Errors (cont.)**

Error #	Explanation
MDV7	An LVS Map Device statement has an invalid new Device component type.
MDV8	An LVS Map Device statement requires a <i>new_subtype</i> parameter.
MDV9	An LVS Map Device statement has an invalid <i>new_subtype</i> parameter.
MDV10	An LVS Map Device statement has an improperly specified <i>new_subtype</i> parameter.
MDV11	An LVS Map Device statement has identical <i>old_subtype</i> and <i>new_subtype</i> parameters.
MDV12	There are duplicate LVS Map Device statements.
MDV13	An LVS Map Device statement has original and new component types specified that are incompatible. Built-in device types must either match (e.g. type C to type C) or be compatible (e.g., type M to ME).

## Net Area Ratio Errors

**Table 7-36. Net Area Ratio Errors**

Error #	Explanation
NAR1	For Net Area Ratio operations, the ACCUMULATE layer and the first denominator layer do not have the same layer of origin, where Stamp and Net Area Ratio ACCUMULATE statements are counted as layer selectors
NAR2	The layer following the ACCUMULATE keyword in a Net Area Ratio operation or the input layer of a Net Area Ratio Print operation is not the output layer of a Net Area Ratio Accumulate operation. The name of the erroneous layer appears in the message.
NAR3	A Net Area Ratio [ACCUMULATE] operation has a superfluous expression specification ( [ ).
NAR4	The secondary keyword PERIMETER ONLY, COUNT ONLY, or SCALE BY has been specified in a Net Area Ratio operation with an expression specification. The name of the erroneous secondary keyword appears in the message.
NAR5	Bad syntax in a Net Area Ratio [ACCUMULATE] expression. The object (number keyword, and so on) where the illegal syntax was detected appears in the message.
NAR6	The layer parameter to an area(), perimeter(), or count() function in a Net Area Ratio [ACCUMULATE] expression is not an input layer in the same operation. The name of the erroneous layer appears in the message.
NAR7	A layer parameter following the RDB keyword in a Net Area Ratio operation is not an input layer in the same operation. The name of the erroneous layer appears in the message.
NAR8	A layer parameter following the RDB keyword in a Net Area Ratio operation appears twice following the RDB keyword. The name of the erroneous layer appears in the message.
NAR9	The BY LAYER keyword following the RDB parameter to a Net Area Ratio operation cannot be specified if there are no layers following the RDB keyword.
NAR10	OVER may not be specified in Net Area Ratio if there is only one input layer.
NAR11	The single-layer form of Net Area Ratio must have an expression specified.
NAR12	The Net Area Ratio Accumulate operation must have an expression specified.
NAR13	The Net Area Ratio operation cannot have INSIDE OF LAYER specified with either ACCUMULATE <i>a</i> layer or RDB.
NAR14	The ACCUMULATE keyword in a Net Area Ratio operation has more than one layer parameter but RDB is not also specified.
NAR15	The Net Area Ratio operation BY NET keyword is specified without INSIDE OF LAYER also being specified.

**Table 7-36. Net Area Ratio Errors (cont.)**

Error #	Explanation
NAR16	The output layer of a Net Area Ratio operation with ACCUMULATE, or the output of a Net Area Ratio Accumulate operation, is a parameter to a Connect operation prior to the NARAC layer's use in a later layer derivation. This is enforced due to internal limitations. The name of the offending layer is placed into the message.

## Numeric Errors

**Table 7-37. Numeric Errors**

Error #	Explanation
NUM1	A numeric operand in a statement or operation is required to be positive or zero but is negative. For example, the value following the TRUNCATE keyword in a Size operation is negative. The erroneous number appears in the error message.
NUM2	A numeric operand in a statement or operation is required to be non-zero but is positive or negative. For example, the value following the SCALE BY keyword in a Net Area Ratio operation is zero. The erroneous numeric appears in the error message.
NUM3	A numeric operand in a statement or operation is required to be positive but is negative or zero. For example, the numeric parameter of the Magnify operation is non-positive. The erroneous numeric appears in the message.
NUM4	A numeric operand in a statement or operation is required to be a non-negative integer but is non-integer or negative. For example, the constraint value(s) for the interact operation are not non-negative integers. Or, for example, the TEXTTYPE parameter of a Layout Text specification statement is not a non-negative integer. The erroneous numeric appears in the message.
NUM5	A numeric operand in a statement or operation is required to be a positive integer but is non-integer, negative, or zero. For example, a parameter of the Precision specification statement is not a positive integer. The erroneous numeric appears in the message.
NUM6	The parameter of the DRC Maximum Vertex or ERC Maximum Vertex specification statements is not an integer greater than or equal to 4. The erroneous numeric appears in the message.
NUM7	A numeric expression, when evaluated, will cause division by zero or a similar exception. This includes: $x \% y$ where $y = 0$ ; $x ^ y$ where $x < 0$ and $y$ is not an integer; $x ^ y$ where $x = 0$ and $y$ is negative.
NUM8	The sizing value of the Size operation, the STEP value of the Size operation, or the numeric parameter of the Deangle operation is not evenly divisible by the database precision. The erroneous numeric appears in the message.
NUM9	A numeric operand in a statement or operation is required to be in the interval [0,1]. The offending numeric is placed into the message.
NUM10	A numeric operand in a statement or operation is required to be greater than or equal to 1. The offending numeric is placed into the message.
NUM11	A numeric operand in a statement or operation is required to be in the interval [0,1). For example, the numeric value of the DRC Tolerance Factor NAR specification statement is not in the interval [0,1). The offending numeric is placed into the message.

**Table 7-37. Numeric Errors (cont.)**

Error #	Explanation
NUM12	A numeric operand in a statement or operation is required to be an integer. The offending numeric is placed into the message.
NUM13	A numeric operand in a statement or operation is required to be on the interval (0,1). The offending numeric is placed into the message.
NUM14	A value on the interval from 0 and less than 100 must be specified.

## Layer Statement Errors

**Table 7-38. Layer Statement Errors**

Error #	Explanation
OLS1	An original layer specification statement lacks a name parameter or has an invalid name parameter.
OLS2	An original layer specification statement has no definition or contains no layer numbers in its definition, even when fully flattened. The name of the erroneous layer statement appears in the error message.
OLS3	The name of an original layer specification statement duplicates that of another original layer statement in the rule file. The erroneous name appears in the error message.
OLS4	An operand within an original layer specification statement is a keyword or is negative, non-integral number, or greater than 65535 (the maximum original layer number), or is a name that is not resolvable as an original layer. The erroneous entity appears in the error message.
OLS5	An original layer statement name is also defined in the Pyxis Process, but the rule file definition does not agree with the Pyxis Process definition. The erroneous name appears in the error message.

## OPC Operation Errors

**Table 7-39. OPC Operation Errors**

Error #	Explanation
OPC1	The second or third input layer of the Opclineend operation is not immediately followed by a by keyword. The name of the erroneous layer appears in the message.
OPC2	The WIDTH, END, SERIF, HEIGHT, or SPACE keyword has been encountered twice in the same parameter group of the Opclineend operation. The erroneous keyword appears in the message.
OPC3	A parameter group of the Opclineend operation lacks one of the three keywords WIDTH, END, or SERIF (including a completely missing parameter group). The name of the missing keyword appears in the message.

**Table 7-39. OPC Operation Errors (cont.)**

Error #	Explanation
OPC4	For two (or more) of the WIDTH/END/SERIF parameter groups of the Opclineend operation, there is intersection of the (possibly infinite) cubic intervals defined with the WIDTH constraint along the x-axis, the HEIGHT constraint along the y-axis, and the SPACE constraint along the z-axis (or vice-versa). If HEIGHT is not explicitly specified, the HEIGHT constraint is treated as $\geq 0$ for the purposes of this check. If SPACE is not explicitly specified in both parameter groups, then there is positive overlap defined along the z-axis for the purpose of this check. If SPACE is not specified in exactly one parameter group, then there is no overlap defined along the z-axis for the purpose of this check.
OPC5	The SPACE, MOVE, WIDTH, LENGTH1, LENGTH2, SQUARE, or OPPOSITE keyword has been encountered twice in the same SPACE/MOVE parameter group of the Opcbias operation. The erroneous keyword appears in the message.
OPC6	A parameter group of the Opcbias operation lacks one of the two keywords SPACE or MOVE (including a completely missing parameter group). The name of the missing keyword appears in the message.
OPC7	For two (or more) of the SPACE/MOVE parameter groups of the Opcbias operation there is intersection of the (possibly infinite) 5-space intervals defined by placing the WIDTH constraint along the x1-axis and the WIDTH2 constraint along the x2-axis, the SPACE constraint along the x3-axis, and the LENGTH1 constraint along the x4-axis, and the LENGTH2 constraint along the x5-axis. Intersection includes abutment and point-to-point contact. If WIDTH (LENGTH1) (LENGTH2) is not specified, then the WIDTH (LENGTH1) (LENGTH2) constraint is treated as $\geq 0$ for the purposes of this check.
OPC8	The first input layer of the Opcbias operation is a derived polygon layer and more than two input layers have been specified.
OPC9	The first, and only, input layer of the Opcbias operation is a derived edge layer.
OPC10	The first input layer of the Opcbias operation is a derived edge layer but is not derived by a sequence of layer selector operations from the second input layer (a derived polygon layer).
OPC11	The optional metric following the SPACE keyword in an Opcbias or Opclineend operation or following the WIDTH keyword in an Opcbias operation is OPPOSITE, SQUARE, or OPPOSITE SYMMETRIC. Currently only OPPOSITE EXTENDED is supported. The name of the unsupported metric appears in the message.
OPC12	An Opcsbar operation lacks a SPACE parameter.
OPC13	The keyword SBWIDTH, SBOFFSET, or CENTER appears in an Opcsbar operation outside of a SPACE parameter group. The erroneous keyword is listed in the message.

**Table 7-39. OPC Operation Errors (cont.)**

Error #	Explanation
OPC14	No SBWIDTH keyword present in an Opcobar operation SPACE parameter group.
OPC15	No SBOFFSET or CENTER keyword following the SBWIDTH keyword in an Opcobar SPACE parameter.
OPC16	The CENTER keyword is present in an Opcobar SPACE parameter group and the constraint following the SPACE keyword is of the form > a, >=a, or !=a.
OPC17	If MINSBOFFSET is not globally specified in an Opcobar operation, then at least one SPACE parameter group must have SBOFFSET specified.
OPC18	For any two Opcobar SPACE parameter groups (SPACE specified, not SPACELAYER), there is an intersection of the (possibly infinite) planar intervals defined with the WIDTH constraint along the x-axis, and the SPACE constraint along the y-axis, or vice-versa. If WIDTH is not explicitly specified, the WIDTH constraint is treated as >= 0 for the purposes of this check.
OPC19	A maximum of one CENTER keyword may be specified in an Opcobar SPACE parameter.
OPC20	A LENGTH2 parameter is specified in an Opcobar SPACE or MOVE parameter group, where the SPACE constraint is of the form > a, >=a, or !=a.
OPC21	If NEGATIVE is globally specified in an Opcobar operation, then no SPACE parameter group may have CENTER specified.
OPC22	Reserved.
OPC23	Reserved.
OPC24	The CORNERFILL keyword must have a <i>layer2</i> parameter in an Opclineend operation.
OPC25	Opclineend requires that a default rule that contains no SPACE keyword be specified for each rule that contains a WIDTH or HEIGHT keyword.
OPC26	An Opcbias statement exceeds 4095 rules.

## General File Input Output Errors

**Table 7-40. General File I/O Errors**

Error #	Explanation
OPEN1	The rule file cannot be opened for read access. The name of the rule file appears in the error message. This message does not report <fname> or <lnum>.
READ1	There is a problem reading the rule file or reading an include file. The name of the problem file appears in the error message.
EVAL1	The file name or path name parameter of any statement or operation is too long, either before or after evaluation of embedded environment variables. The problem name appears in the message.
EVAL2	The results database filename in a DRC Results Database or DRC Check Map specification statement begins with the character sequence PIPE but is otherwise empty or blank.

## Rule Check Statement Errors

**Table 7-41. Rule Check Statement Errors**

Error #	Explanation
OUT1	There are no free-standing layer operations within a rule check statement; that is, there will be no output generated from the rule check statement. The name of the erroneous rule check statement appears in the error message.
OUT2	The name of a rule check statement duplicates that of another rule check statement in the rule file. The erroneous rule check statement name appears in the error message.

## Parasitic Extraction Errors

**Table 7-42. Parasitic Extraction Errors**

Error #	Explanation
PEX1	More than one capacitance order operation for the given connectivity set was encountered in the rule file.

**Table 7-42. Parasitic Extraction Errors (cont.)**

Error #	Explanation
PEX2	<p>Multiple specification of any of the following:</p> <p>Capacitance Intrinsic A      Capacitance Intrinsic Plate A      Capacitance Intrinsic Fringe A      Capacitance Intrinsic A INSIDE OF C      Capacitance Intrinsic Plate A INSIDE OF C      Capacitance Intrinsic Fringe A INSIDE OF C      Capacitance Intrinsic Fringe A INSIDE OF C D      Capacitance Intrinsic A B (A and B in that order)      Capacitance Intrinsic Plate A B (A and B in that order)      Capacitance Intrinsic Fringe A B (A and B in that order)      Capacitance Intrinsic A B INSIDE OF C (A and B in that order)      Capacitance Intrinsic Plate A B INSIDE OF C (A and B in that order)      Capacitance Intrinsic Fringe A B INSIDE OF C (A and B in that order)      Capacitance Intrinsic Fringe A B INSIDE OF C D (A and B in that order)      Capacitance Crossover A B (A and B in either order)      Capacitance Crossover Plate A B (A and B in either order)      Capacitance Crossover Fringe A B (A and B in that order)      Capacitance Crossover A B INSIDE OF C (A and B in either order)      Capacitance Crossover Plate A B INSIDE OF C (A and B in either order)      Capacitance Crossover Fringe A B INSIDE OF C (A and B in that order)      Capacitance Crossover A B INSIDE OF C D (A and B in either order)      Capacitance Crossover Plate A B INSIDE OF C D (A and B in either order)      Capacitance Crossover Fringe A B INSIDE OF C D (A and B in that order)      Capacitance Nearbody A [WITH SHIELD S]      Capacitance Nearbody A B [WITH SHIELD S] (A and B in either order)      Capacitance Nearbody A INSIDE OF C      Capacitance Nearbody A INSIDE OF C D      Capacitance Nearbody A B INSIDE OF C (A and B in either order)      Capacitance Nearbody A B INSIDE OF C D (A and B in either order)      Capacitance Connection A B (A and B in that order)      Resistance Connection A B (A and B in that order)      Resistance Sheet A</p> <p>The name of the erroneous layer appears in the error message.</p>
PEX3	A layer parameter to a Capacitance Crossover operation or the first or INSIDE OF layer parameter to a Capacitance Intrinsic operation or any layer parameter to a Capacitance Nearbody operation with INSIDE OF specified does not appear as a parameter to a Capacitance Order operation. The name of the erroneous layer appears in the error message.

**Table 7-42. Parasitic Extraction Errors (cont.)**

Error #	Explanation
PEX4	<p>A base layer in a Capacitance Intrinsic operation cannot appear as:</p> <ol style="list-style-type: none"> <li>1. Any layer in a Capacitance Connection or resistance extraction operation.</li> <li>2. Any layer in a Capacitance Crossover or Capacitance Nearbody operation.</li> <li>3. The first layer in a Capacitance Intrinsic operation.</li> </ol> <p>The name of the erroneous layer appears in the error message.</p>
PEX5	<p>This error has six possible sources:</p> <ol style="list-style-type: none"> <li>1. The layer parameter to a Resistance Sheet operation does not appear as a connector layer in any Connect BY operation.</li> <li>2. The layer parameter to a Resistance Sheet operation appears as a contact layer in a Connect BY operation.</li> <li>3. The first layer parameter to a Resistance or Capacitance Connection operation does not appear as the first connector layer in any Connect BY operation.</li> <li>4. The second layer parameter to a Resistance or Capacitance Connection operation does not appear as a connector layer in a Connect BY operation where the first layer parameter is the first connector layer.</li> <li>5. The layer parameter to a Resistance or Capacitance Connection operation appears as a contact layer in some Connect BY operations.</li> <li>6. The layer parameter in the Capacitance Order statement includes contact or via layers.</li> </ol> <p>The name of the erroneous layer appears in the error message.</p>
PEX6	A parasitic extraction operation is missing its property specification. The name of the erroneous operation appears in the message.
PEX7	A parasitic extraction operation has a superfluous property specification ( [ ).

**Table 7-42. Parasitic Extraction Errors (cont.)**

Error #	Explanation
PEX8	A layer in a Capacitance Intrinsic or Capacitance Crossover operation is not properly ordered within the Capacitance Order operation. Specifically (where $L1 < L2$ means that $L1$ is to the left of $L2$ in the Capacitance Order operation, and $L1 > L2$ means that $L1$ is to the right of $L2$ in the Capacitance Order operation) any of the following can cause this error: <ol style="list-style-type: none"><li>1. Capacitance Intrinsic Plate A [B] INSIDE OF C, where <math>A &gt; C</math>.</li><li>2. Capacitance Intrinsic [PLATE   FRINGE] A B INSIDE OF C, where if B is in the capacitance order operation, <math>B &gt; C</math>.</li><li>3. Capacitance Intrinsic Fringe A [B] INSIDE OF C D where <math>A &lt; C</math> or <math>A &gt; D</math>.</li><li>4. Capacitance Crossover [PLATE] A B INSIDE OF C, where <math>C &lt; A</math> and <math>C &gt; B</math>, or where <math>C &lt; B</math> and <math>C &gt; A</math>.</li><li>5. Capacitance Crossover [PLATE] A B INSIDE OF C D where <math>C &gt; A</math>, or where <math>C &gt; B</math>, or where <math>D &lt; A</math>, or where <math>D &lt; B</math>.</li><li>6. Capacitance Crossover Fringe A B INSIDE OF C, where neither <math>C &lt; A</math> nor <math>C &gt; A</math>.</li><li>7. Capacitance Crossover Fringe A B INSIDE OF C D where <math>C &gt; A</math> or <math>D &lt; A</math>.</li><li>8. Capacitance Nearbody A INSIDE OF C where neither <math>C &lt; A</math> nor <math>C &gt; A</math>.</li><li>9. Capacitance Nearbody A INSIDE OF C D where <math>C &gt; A</math> or <math>D &lt; A</math>.</li><li>10. Capacitance Nearbody A B INSIDE OF C, where <math>C &lt; A</math> and <math>C &gt; B</math>, or where <math>C &lt; B</math> and <math>C &gt; A</math>.</li><li>11. Capacitance Nearbody A B INSIDE OF C D where <math>C &gt; A</math>, or where <math>C &gt; B</math>, or where <math>D &lt; A</math>, or where <math>D &lt; B</math>.</li></ol>

## PEX Statement Errors

**Table 7-43. PEX Statement Errors**

Error #	Explanation
PSS1	Multiple PEX statements of the same type given in the same connectivity mode, or multiple specifications of PEX Tolerance Distributed or PEX Reduce Coupled. The primary keyword of the erroneous statement appears in the error message.
PSS2	A PEX statement is missing a parameter. This error has the following sources: 1. No <name> parameter in a PEX Extract Include or PEX Extract Exclude statement. 2. Neither <filename> or None specified in a PEX Report Lumped, PEX Report Distributed, PEX Netlist Simple, PEX Netlist ADMS, or PEX Netlist statement. 3. Neither R or RC specified in a PEX Tolerance Distributed specification statement. 4. No <tvalue> specified in a PEX Threshold statement. 5. No C specified in a PEX Reduce Lumped specification statement. 6. No <value> specified in a PEX Reduce Coupled specification statement. 7. Neither GLOBAL nor FILE specified in a PEX Reduce Coupled specification statement. 8. The <layer1> layer in a Capacitance Ignore is not part of a Capacitance Alias layer set.  The primary keyword of the erroneous statement appears in the error message.
PSS3	The type secondary keyword ASCII is not specified for a PEX Report Distributed statement when None was not specified.
PSS4 - PSS7	Deprecated.
PSS8	The type secondary keyword HSPICE is not specified for a PEX Netlist Simple or PEX Netlist statement when None was not specified.
PSS9	The system secondary keyword ID, SOURCE, or LAYOUT in a PEX Netlist, PEX Netlist Simple PEX Report Lumped, or PEX Report Distributed statement when None was not specified.
PSS10	The PEX statement is missing the WINDOW keyword.
PSS11	The PEX Layer statement is repeated.

## PEX Property Computation Errors

**Table 7-44. PEX Property Computation Errors**

Error #	Explanation
PXR1	Too few numbers in numeric parameter set. The erroneous statement appears in the error message.
PXR2	Too many numbers in numeric parameter set. The erroneous numeric appears in the error message.
PXR3	Only numeric parameter set is allowed for this property specification. The erroneous property specification appears in the error message.
PXR4	Missing property identifier. The erroneous keyword appears in the error message.
PXR5	Property identifier must be C. The erroneous property specification appears in the error message.
PXR6	Property identifier must be R. The erroneous property specification appears in the error message.
PXR7	This property was not assigned a value in all cases. The erroneous property specification appears in the error message.
PXR8	Default property was not assigned a value in all cases. The erroneous property specification appears in the error message.
PXR9	This property was never assigned a value. The erroneous property specification appears in the error message.
PXR10	Default property was never assigned a value. The erroneous property specification appears in the error message.
PXR11	Missing statement. The erroneous statement appears in the error message.
PXR12	Unexpected string found where an expression representing a numeric value was expected. The erroneous expression appears in the error message.
PXR13	Unexpected string found where a comma was expected. The erroneous string appears in the error message.
PXR14	Unexpected string found where a right brace } was expected. The erroneous string appears in the error message.
PXR15	Unexpected string found where a right bracket ] was expected. The erroneous string appears in the error message.
PXR16	Unexpected string found where a left parenthesis ( was expected. The erroneous string appears in the error message.
PXR17	Unexpected string found where a right parenthesis ) was expected. The erroneous string appears in the error message.

**Table 7-44. PEX Property Computation Errors (cont.)**

Error #	Explanation
PXR18	Unexpected string found where a relational operator such as <= was expected. The erroneous string appears in the error message.
PXR19	This process variable does not have a numeric value. The erroneous variable appears in the error message.
PXR20	Unexpected string found where a property or variable name was expected. The erroneous item appears in the error message.
PXR21	This variable was used on the right before it was given a value on the left. The erroneous variable appears in the error message.
PXR22	This variable was used on the right before it was unconditionally initialized. The erroneous variable appears in the error message.
PXR23	Assignment to this process variable is not allowed. The erroneous variable appears in the error message.
PXR24	MAX_DISTANCE variable is required for this property specification. The erroneous property specification appears in the error message.
PXR25	MAX_DISTANCE variable is not initialized to a positive number. The erroneous variable appears in the error message.
PXR26-30	Not used.
PXR31	MAX_WIDTH variable is required for this property specification. The erroneous property specification appears in the error message.
PXR32	MAX_WIDTH variable is not initialized to a positive number. The erroneous variable appears in the error message.
PXR33	MAX_ENCLOSE variable is required for this property specification. The erroneous property specification appears in the error message.
PXR34	MAX_ENCLOSE variable is not initialized to a positive number. The erroneous variable appears in the error message.

## RDB Conflict Errors

**Table 7-45. RDB Conflict Errors**

Error #	Explanation
RDB1	The file name parameter of the RDB keyword in a Density operation duplicates the file name in a DRC Results Database or DRC Check Map specification statement. The erroneous file name is placed into the message.
RDB2	The file name parameter of the RDB keyword in a Net Area Ratio operation duplicates the file name in a DRC Results Database or DRC Check Map specification statement. The erroneous file name is placed into the message.
RDB3	The file name parameter of the Layout Input Exception RDB specification statement duplicates the file name in a DRC Results Database or DRC Check Map specification statement. The erroneous file name is placed into the message.
RDB4	The RDB file name parameter of a DFM operation or specification statement duplicates the file name in a DRC Results Database or DRC Check Map specification statement. The erroneous file name is placed into the message.

## Layer and Variable Resolution Errors

**Table 7-46. Layer and Variable Resolution Errors**

Error #	Explanation
RES1	A variable name used in an operation or statement cannot be resolved as the name of a Variable specification statement or as an Pyxis process variable. The erroneous variable name is placed in the error message.
RES2	A layer name parameter for an operation or DRC Print Area, DRC Print Perimeter, or LVS Softchk specification statement is neither the name of a derived layer nor the name of an original layer. The erroneous layer name appears in the error message.
RES3	A layer name parameter for the Text, Layout Text, Layout Polygon, Text Layer, Layer Resolution, Layout Base Layer, Layout Top Layer, Layout Window Layer, Layout Windel Layer, DRC Select Check By Layer, DRC Unselect Check By Layer, DRC Map Text Layer, Port Layer Text, Port Layer Polygon, or Polygon specification statement; or the text layer name in a With Text, Not With Text, or Expand Text operation, or any layer name parameter of the Extent Drawn operation, or the Text Model Layer name parameter for the Device operation, is not the name of an original layer. The erroneous layer name appears in the message.

**Table 7-46. Layer and Variable Resolution Errors (cont.)**

Error #	Explanation
RES4	An original layer number parameter for a Text Layer, Text, Layout Text, Layout Polygon, Layout Window Layer, Layout Windel Layer, DRC Map Text Layer, Port Layer Text, Port Layer Polygon, Polygon, DRC Print Area, DRC Print Perimeter, or LVS Softchk or PEX Layer specification statement, or the target layer of a Layer Map specification statement is non-integral, negative, or greater than 65535 (the maximum allowable original layer number). The erroneous number appears in the error message.
RES5	A variable name used in an operation or specification statement is resolvable, but has a non-numeric type. The erroneous variable name appears in the error message.
RES6	A variable name used in an operation or specification statement is resolvable, but has a non-string type. The erroneous variable name appears in the message.
RES7	A variable name used in an operation or specification statement is resolvable, but has a non-numeric or non-string type. The erroneous variable name appears in the message.
RES8	A string variable name used in an operation or specification statement may not have more than one value in its definition. The errant variable name is placed into the message.

## LVS Recognize Gates Tolerance Statement Errors

**Table 7-47. LVS Recognize Gates Tolerance Statement Errors**

Error #	Explanation
RGT1	An LVS Recognize Gates Tolerance statement contains a missing or invalid Device component name.
RGT2	An LVS Recognize Gates Tolerance statement contains an improper Device model (subtype) name.
RGT3	An LVS Recognize Gates Tolerance statement contains a missing or invalid Device property name.
RGT4	An LVS Recognize Gates Tolerance statement contains an invalid SPICE value for a specified property.
RGT5	An LVS Recognize Gates Tolerance statement contains a missing or invalid tolerance parameter.
RGT6	There are duplicate LVS Recognize Gates Tolerance statements.
RGT7	An LVS Recognize Gates Tolerance statement contains a disallowed SPICE value for a specified property.
RGT8	An LVS Recognize Gates Tolerance statement references a Device component that is not defined.
RGT9	An LVS Recognize Gates Tolerance statement references a property that is not computed for all Device definition to which the statement applies.
RGT10	An LVS Recognize Gates Tolerance statement references a property of incorrect type in at least one Device definition to which the statement applies.

## LVS Split Gate Ratio Statement Errors

**Table 7-48. LVS Split Gate Ratio Statement Errors**

Error #	Explanation
SGR1	A LVS Split Gate Ratio specification statement lacks an element name parameter or has an invalid element name parameter.
SGR2	A model name within a LVS Split Gate Ratio specification statement is not of the form '(' <name> ')'. The element name of the erroneous statement appears in the message. (Not currently used).
SGR3	A LVS Split Gate Ratio specification statement does not have a property specified or the specified property is not a name. The element name of the erroneous statement appears in the message.

**Table 7-48. LVS Split Gate Ratio Statement Errors (cont.)**

Error #	Explanation
SGR4	A spice value within a LVS Split Gate Ratio specification statement is not of the form “(” <name> “)”, or the <name> is invalid, that is, is not w, W, l, L, r, R, c, C, a, A, p, P. The property name associated with the spice value appears in the message.
SGR5	A LVS Split Gate Ratio specification statement does not have a tolerance specified or the specified tolerance is not a number. The element name of the erroneous statement appears in the message.
SGR6	A LVS Split Gate Ratio specification statement shares its (<element name>,<property>) with a previous LVS Split Gate Ratio specification statement in the same set (direct source, direct layout, mask source, mask layout). The element name of the erroneous statement appears in the message.
SGR7	A mask mode LVS Split Gate Ratio specification statement property has a spice value specified. This error results from consistency checking against rule file device definitions. The name of the erroneous property appears in the message.
SGR8	A mask mode LVS Split Gate Ratio specification statement has no corresponding device definition in the rule file. This error results from consistency checking against rule file device definitions. The element name of the erroneous statement appears in the message.
SGR9	A property in a mask mode LVS Split Gate Ratio specification statement is not computed in all corresponding rule file device definitions. This error results from consistency checking against rule file device definitions. The erroneous property name appears in the message.
SGR10	A property in a mask mode LVS Split Gate Ratio specification statement is computed in some corresponding rule file device definition with a type different from the type (string or numeric) specified in the statement. This error results from consistency checking against rule file device definitions. The erroneous property name appears in the message.

## General Specification Statement Errors

**Table 7-49. General Specification Statement Errors**

Error #	Explanation
SPC1	A specification statement is restricted to a single occurrence in the rule file, yet a second one was encountered. The primary keyword of the erroneous specification statement appears in the message.
SPC2	A specification statement is missing a parameter. The primary keyword of the erroneous specification statement appears in the message.
SPC3	The numeric parameter of the Precision specification statement differs from the precision specified by the Pyxis process.
SPC4	A Text specification statement has a <layer> parameter but lacks a <name> parameter.
SPC5	A <name> parameter in a DRC Select Check, DRC Unselect Check, ERC Select Check, or ERC Unselect Check specification statement is not the name of any nmDRC RuleCheck statement or RuleCheck group. The erroneous name appears in the message.
SPC6	A <name> parameter in a LVS Heap Directory specification statement duplicates that of a previous LVS Heap Directory specification statement (case-sensitive comparison). The erroneous name appears in the message.
SPC7	The coordinate list in a Polygon or Layout Polygon specification statement (defined as all parameters except the last, which is assumed to represent the layer) is too short (less than 4 numerics), or has an odd number of numerics, or is an invalid orthogonal rectangle, or represents a self-intersecting polygon.
SPC8	Deprecated.
SPC9	The coordinate list in a Layout Window or Layout Windel specification statement is too short (less than four numerical values), or has an odd number of numerics, or is an invalid orthogonal rectangle, or represents a self-intersecting polygon.
SPC10	A Layer Resolution specification statement is specified more than once for the same original layer name, or the original layer name appears more than once in the same Layer Resolution statement. The name of the erroneous layer is placed in the message.
SPC11	The layer parameter to a DRC Print Area or LVS Softchk specification statement would not otherwise be generated. The name of the erroneous layer is placed in the message.
SPC12	The layer parameter to an LVS Softchk specification statement is not the lower layer in a Sconnect operation. The name of the erroneous layer is placed in the message.
SPC13	Layout Rename Cell specification statement is specified twice for the same source cell name. The name of the erroneous layer is placed in the message.

**Table 7-49. General Specification Statement Errors (cont.)**

<b>Error #</b>	<b>Explanation</b>
SPC14	A pattern matching argument for the Layout Rename Text specification statement is invalid. The name of the erroneous argument appears in the message.
SPC15	A Layout Input Exception Severity specification statement has been specified twice for the same exception name. The exception name of the errant statement is placed into the message.
SPC16	A Layout Input Exception Severity specification statement has a severity value specified which is not permissible for the associated exception name. The exception name of the errant statement is placed into the message.
SPC17	The Layout Magnify specification statement has a PLACE parameter and a Layout Path or Layout Path2 specification statement contains a MAG keyword. PLACE is not supported in this scenario.
SPC18	RDB is specified in a Layout Input Exception Severity specification statement, but the exception type does not support results database output. The name of the exception is placed into the message.
SPC19	RDB is specified in a Layout Input Exception Severity specification statement, but the exception value does not support results database output. The name of the exception is placed into the message.
SPC20	RDB is specified in a Layout Input Exception Severity specification statement, but there is no Layout Input Exception RDB statement in the rule file.

## LVS Spice Rename Parameter Statement Errors

**Table 7-50. LVS Spice Rename Parameter Statement Errors**

Error #	Explanation
SPI1	An LVS Spice Rename Parameter statement has an improper subtype name.
SPI2	An LVS Spice Rename Parameter statement with more than one component type has an improper subtype name.
SPI3	An LVS Spice Rename Parameter statement has a parameter name of a disallowed form. SVRF keywords and numeric parameters are not allowed.
SPI4	An LVS Spice Rename Parameter rule conflicts with a previous one.
SPI5	An LVS Spice Rename Parameter statement has an element name specified that is either invalid or does not appear in a Device definition.

## Syntax Errors

**Table 7-51. Syntax Errors**

Error #	Explanation
SYN1	A section of the rule file cannot be recognized as an operation since it lacks a primary keyword.
SYN2	A left bracket ( [ ) is not paired with a right bracket ( ] ), or vice versa, or left brackets are nested.
SYN3	A layer definition statement designator (=) or RuleCheck statement designator ( { ) is not preceded by a name. The erroneous statement designator appears in the message.
SYN4	Style comment /* ... */ is nested, unterminated, or unmatched.
SYN5	Invalid or incomplete numeric expression near: <object> Bad syntax in a numeric expression. The object (number, keyword, and so on) where the illegal syntax was detected appears in the message.
SYN6	A left parenthesis ( is not paired with a right parenthesis ), or vice versa. This is reported for any unmatched parentheses outside of brackets [ ].
SYN7	Names beginning with tmp< are reserved. Names of the form tmp<... cannot be used since they are used internally for unwinding implicit layer definitions. The < character can only occur in string constants.
SYN8	Unmatched braces ( { } ) outside of brackets ( [ ] ).

## Trace Property Statement Errors

**Table 7-52. Trace Property Statement Errors**

Error #	Explanation
TRP1	A Trace Property statement lacks an element name parameter or has an invalid element name parameter.
TRP2	The model name in a Trace Property statement is not of the form “(”<name>“)”. The element name of the erroneous Trace Property statement appears in the error message.
TRP3	A Trace Property statement does not have a source property specified or the specified source property is not a name. The element name of the erroneous Trace Property statement appears in the error message.
TRP4	A Trace Property statement does not have a layout property specified or the specified layout property is not a name. The element name of the erroneous Trace Property statement appears in the error message.
TRP5	A SPICE value in a Trace Property statement is not of the form “(”<name>“)” or <name> is invalid. The property name associated with the SPICE value appears in the error message.
TRP6	A Trace Property statement with a source or layout property SPICE value has a STRING keyword specified. The element name of the erroneous Trace Property statement appears in the error message.
TRP7	A Mask mode numeric Trace Property statement shares its (<element name><model name><source property>) set with a previously defined Mask mode numeric Trace Property statement. The element name of the erroneous Trace Property statement appears in the error message.
TRP8	A Mask mode string Trace Property statement shares its (<element name><model name><source property>) set with a previously defined Mask mode string Trace Property statement. The element name of the erroneous Trace Property statement appears in the error message.
TRP9	A Direct mode numeric Trace Property statement shares its (<element name><model name><source property>) set with a previously defined Direct mode numeric Trace Property statement. The element name of the erroneous Trace Property statement appears in the error message.
TRP10	A Direct mode string Trace Property statement shares its (<element name><model name><source property>) set with a previously defined Direct mode string Trace Property statement. The element name of the erroneous Trace Property statement appears in the error message.
TRP11	A Direct (Mask) mode numeric Trace Property statement shares its (<element name><model name><source property>) set with a previously defined (Mask) Direct mode numeric Trace Property statement. The element name of the erroneous Trace Property statement appears in the error message.

**Table 7-52. Trace Property Statement Errors (cont.)**

Error #	Explanation
TRP12	A Mask mode Trace Property statement has a SPICE value specified. The name of the erroneous property appears in the error message.
TRP13	A Mask mode Trace Property statement has no corresponding device definition in the rule file. The element name of the erroneous Trace Property statement appears in the error message.
TRP14	A property in a Mask mode Trace Property statement is not computed in all corresponding rule file device definitions. The name of the erroneous property appears in the error message.
TRP15	A property in a Mask mode Trace Property statement is computed in some corresponding rule file device definitions with a type different from the type (string or numeric) specified in the statement. The erroneous property name appears in the error message.
TRP16	A name following the tolerance keyword within a Trace Property specification statement is not of the form '( <name> ')'. The element name of the errant Trace Property specification statement is placed into the message.
TRP17	A name that is not declared in a PROPERTY statement is in a location where a PROPERTY name is expected. This occurs within a user-defined program in a Trace Property statement.
TRP18	A name that appears in a PROPERTY statement appears next to an = sign. PROPERTY names cannot be in such assignment statements. This occurs within a user-defined program in a Trace Property statement.
TRP19	A variable name is missing from a location where a variable is expected within a user-defined program in a Trace Property statement.
TRP20	A name that is declared in a PROPERTY STRING statement is missing from an expected location. This occurs within a user-defined program in a Trace Property statement.
TRP21	A scalar string expression is missing from an expected location. This occurs within a user-defined program in a Trace Property statement.
TRP22	There is a duplicate Trace Property statement with a user-defined program.
TRP23	A Trace Property statement has a mixture of ordinary syntax and user-defined program syntax.
TRP24	A Trace Property statement using built-in property tracing for a given element(model) is present when a Trace Property statement with a user-defined program specifies the same element. The name of the element is reported.
TRP25	A Trace Property statement with a user-defined program for a given element(model) is present when a Trace Property statement using built-in property tracing specifies the same element. The name of the element is reported.

## Layer Type Errors

**Table 7-53. Layer Type Errors**

Error #	Explanation
TYP1	The layer parameter for any one-layer Boolean operation, a Pins, Ports, Topex, or (Not) Inside Cell operation, or the first layer parameter for an Attach operation is not an original layer. The name of the erroneous layer appears in the error message.
TYP2	An original layer or derived polygon layer appeared as a layer parameter whose type must be a derived edge layer. The name of the erroneous layer appears in the error message.
TYP3	A derived edge layer appeared as a layer parameter whose type must not be a derived edge layer. The name of the erroneous layer appears in the error message.
TYP4	A derived error layer appeared as a parameter to any operation or as a parameter to a DRC Print Area, DRC Print Perimeter, or LVS Softchk specification statement. The name of the erroneous layer appears in the error message.
TYP5	The input layer type must be derived polygon.
TYP6	The input layer type must be derived error.
TYP7	The input layer type must not be derived polygon.
TYP8	The input layer must not be derived outside the scope of the rule check.
TYP9	The input layer must be local to the rule check scope.
TYP10	The input layer must not be an original layer.

## SVRF Error Statement

**Table 7-54. SVRF Error Statement**

Error #	Explanation
USER1	An SVRF Error specification statement is encountered in the rule file during compilation.

## Variable Statement Errors

**Table 7-55. Variable Statement Errors**

Error #	Explanation
VAR1	A Variable specification statement lacks a name parameter or has an invalid name parameter.

**Table 7-55. Variable Statement Errors (cont.)**

Error #	Explanation
VAR2	The value in a Variable specification statement is missing or is not a number, the keyword ENVIRONMENT, or a string constant. The Variable specification statement name appears in the message.
VAR3	The name of a Variable specification statement duplicates that of another Variable specification statement in the rule file. The erroneous name appears in the message.
VAR4	A Variable specification statement name is also defined in the IC Station process, but the rule file definition does not agree with the IC Station process definition. The erroneous name appears in the message.

## SVRF Version Statement Errors

**Table 7-56. SVRF Version Statement**

Error #	Explanation
VER1	The parameter of a SVRF Version specification statement is not recognizable as a valid Calibre version string. The string is not of the form vDDDDPD_DD...DPDD...D where v is literal, D is any decimal digit, and P is a decimal point.
VER2	The Calibre version predates that specified in an SVRF Version specification statement.

## Runtime Messages

This section covers runtime error and warning messages produced by Calibre nmDRC, DFM, FRACTURE, MDP, OPC, nmLVS, xRC, and ICverify applications. All rule file compilation errors are discussed earlier in this chapter.

Of interest are errors, which are always fatal; exceptions, which might be fatal or non-fatal; warnings, which are non-fatal; and notes, which are informational.

In Calibre, the [Layout Input Exception Severity](#) statement allows freedom in configuring the handling of exceptions. Such handling can range from quietly ignoring an exception, to specifying it as a fatal error. [Layout Error On Input](#) (which is superseded by Layout Input Exception severity) allows you to convert some exceptions into fatal errors.

Calibre nmDRC-F and nmDRC-H return status 1 if a fatal error occurred, else status 0.

Output from the ICrules CHEck DRc command consists of errors (fatal to the command), warnings, and notes. In the first case an error condition has occurred and the command will produce no side effects. Exceptions encountered during the loading of placement data for the CHEck DRc command are handled by IC Station, not ICrules. Some warnings generated by Calibre nmDRC can create a possible data integrity issue since objects causing these warnings may be excluded from processing.

## Calibre nmDRC-H Warnings

**Table 7-57. Calibre nmDRC-H Warnings**

Explanation
These warnings are reported by Calibre nmDRC-H:
<p><b>WARNING:</b> Creating a FLAT COPY of layer &lt;layer&gt;. Reported whenever a temporary flat copy of a layer is created in Calibre nmDRC-H.</p>
<p><b>WARNING:</b> Layer &lt;layer&gt; CONVERTED to an exclusive FLAT instantiation. Reported whenever a layer is converted from an exclusive hierarchical instantiation or dual instantiation to an exclusive flat instantiation in Calibre nmDRC-H.</p>
<p><b>WARNING:</b> Layer &lt;layer&gt; CONVERTED to a dual FLAT/hierarchical instantiation. Reported whenever a layer is converted from an exclusive hierarchical or flat instantiation to a dual instantiation in Calibre nmDRC-H.</p>
<p><b>WARNING:</b> Layer &lt;layer&gt; CONVERTED to an exclusive HIERARCHICAL instantiation. Reported whenever a layer is converted from a dual instantiation or exclusive flat instantiation to an exclusive hierarchical instantiation in Calibre nmDRC-H.</p>
<p><b>WARNING:</b> Layer &lt;layer&gt; CREATED with an exclusive FLAT instantiation. Reported whenever a layer is created with an exclusive flat instantiation in Calibre nmDRC-H.</p>
<p><b>WARNING:</b> Layer &lt;layer&gt; CREATED with a dual FLAT/hierarchical instantiation. Reported whenever a layer is created with a dual instantiation in Calibre nmDRC-H.</p>
<p><b>WARNING:</b> SPACE filter not supported - both SPACE and NOTCH checked. A derived edge layer with an exclusive flat instantiation is input to a one-layer External operation (with one input layer) with a SPACE filter specified in Calibre nmDRC-H. Or a layer with an exclusive flat instantiation is input to a one-layer External operation (with two input layers) with a SPACE filter specified in Calibre nmDRC-H. Both NOTCH and SPACE are checked in that event.</p>
<p><b>WARNING:</b> NOTCH filter not supported - both SPACE and NOTCH checked. A derived edge layer with an exclusive flat instantiation is input to a one-layer External operation (with one input layer) with a NOTCH filter specified in Calibre nmDRC-H. Or a layer with an exclusive flat instantiation is input to a one-layer External operation (with two input layers) with a NOTCH filter specified in Calibre nmDRC-H. Both NOTCH and SPACE are checked in that event.</p>

**Table 7-57. Calibre nmDRC-H Warnings (cont.)**

Explanation
<p>WARNING: One-layer INTERNAL check performed without regard to polygon numbers.</p> <p>A derived edge layer with an exclusive flat instantiation is input to a one-layer Internal operation (with one input layer) in Calibre nmDRC-H. Or a layer with an exclusive flat instantiation is input to a one-layer Internal operation (with two input layers) in Calibre nmDRC-H. Checking is performed without regard to polygon numbers in that event.</p>
<p>WARNING: PROJECTING constraint only supported with PARALLEL ONLY - setting PARALLEL ONLY.</p> <p>A dimensional check operation has a constrained PROJECTING filter in Calibre nmDRC-H but the PARALLEL ONLY filter is not also specified. The operation is executed as if PARALLEL ONLY was specified.</p>
<p>WARNING: Output operation &lt;name&gt; abbreviated due to high probability of exceeding maximum result limit.</p> <p>An External, Enclosure, or Internal output-only operation has been prematurely completed due to the generation of such a large number of nmDRC results that there is a high probability of exceeding the maximum result limit for this operation. The name of the operation appears in the message.</p>
<p>WARNING: Hierarchical OFFGRID checking with unequal X and Y resolutions checks rotated/reflected placements flat.</p> <p>Offgrid checking for the Flag Offgrid specification statement or the Drawn Offgrid or Offgrid operation is processing a resolution with unequal x- and y-values. In this case, rotated and/or reflected placements will be checked flat, which may be slow.</p>
<p>WARNING: OFFGRID placement of &lt;pcell&gt; in &lt;cell&gt; at &lt;point&gt; will require FLAT offgrid/SNAP processing for all placements of this cell.</p> <p>An offgrid placement is encountered during offgrid checking for the Flag Offgrid specification statement or the Drawn Offgrid or Offgrid operation, or during snapping for the Snap Offgrid specification statement or the snap operation. This will cause all placements of this cell to be processed flat, which may be slow. The name &lt;pcell&gt; of the placement, its containing cell name &lt;cell&gt;, and the offgrid basepoint &lt;point&gt; (in &lt;cell&gt; coordinate space) appears in the message.</p>

## Calibre File Input Output Errors

**Table 7-58. Calibre File I/O Errors**

Explanation
The following fatal errors are reported by all Calibre applications:
ERROR: Failure to open input file <filename> for read access. The layout stream file parameter of the Layout Path or Layout Path2 specification statement could not be opened for read access.
ERROR: Failure to read the input file <filename> at record offset <number>. A problem was encountered reading the specified stream file. The record offset where the read problem occurred appears in the message.
ERROR: Failure to read the input file <filename> at file offset <number>. The layout database format is OASIS and a problem was encountered reading the specified input file. The file offset where the read problem occurred appears in the message.
ERROR: Failure to seek to file offset <number> in input file <filename>. The layout database format is strict-mode OASIS and a problem was encountered seeking within the specified input file. The file offset where the seek problem occurred is placed into the message.
ERROR: Cannot open LAYOUT PLACE CELL file <filename> for output. The filename argument (<filename>) of the Layout Place Cell specification statement cannot be opened for write access.
The following fatal errors are reported by Calibre nmDRC applications:
ERROR: Cannot open DRC results database file <filename> for output. The specified DRC results database file cannot be opened for write access.

## Calibre Precision Exceptions

**Table 7-59. Calibre Precision Exceptions**

Comment	Explanation
	<p>The following exceptions are reported by all Calibre applications:</p> <p>Warnings preceded by “†” can have their severities controlled by <a href="#">Layout Input Exception Severity</a>. Warnings preceded by “*” are fatal if <a href="#">Layout Error On Input YES</a> is specified.</p>
†*	<p>EXCEPTION: Layout Precision &lt;number&gt; is not consistent with database precision &lt;number&gt; in input file &lt;filename&gt;.</p> <p>The Layout Precision specification statement value or its local override following the PREC keyword in the Layout Path(2) specification statement is not equal to that specified in the (corresponding) GDSII or OASIS input layout database file.</p> <p>Exception Name: PRECISION_LAYOUT</p>

**Table 7-59. Calibre Precision Exceptions (cont.)**

<b>Comment</b>	<b>Explanation</b>
†*	EXCEPTION: Rule file precision <number> is not consistent with database precision <number> in input file <filename>. The database precision specified in the rule file (or its default) is not equal to that specified in the input layout database file. Exception Name: PRECISION_RULE_FILE
†*	EXCEPTION: Rule file UNIT LENGTH <number> is not consistent with physical precision <number> in input file <filename>. The unit length specified in the rule file (or its default) is not equal to that specified in the input layout database file. The rule file unit length is always used. Exception Name: METRIC_RULE_FILE
†*	EXCEPTION: Database precision <number> in input file <name> is not consistent with database precision <number> in input file <name>. Multiple layout files are being input and the database precision in the first file is inconsistent with that in a subsequent file. Exception Name: PRECISION_INPUT_FILE
†*	EXCEPTION: Physical precision <number> in input file <name> is not consistent with physical precision <number> in file <name>. Multiple files are being input and the unit length in the first file is inconsistent with that in a subsequent file. Exception Name: METRIC_INPUT_FILE

## Calibre Cell and Placement Exceptions

**Table 7-60. Calibre Cell and Placement Exceptions**

Comment	Explanation
	<p>The following exceptions are reported by all Calibre applications:</p> <p>Warnings preceded by “†” can have their severities controlled by <a href="#">Layout Input Exception Severity</a>. Warnings preceded by “*” are fatal if <a href="#">Layout Error On Input YES</a> is specified.</p>
†	<p>EXCEPTION: Excluded cell name &lt;name&gt; not within the input layout database.  A cell name in a rule file Exclude Cell specification statement is not located in the input layout database.</p> <p>Exception name: EXCLUDE_CELL_NAME</p>
†	<p>EXCEPTION: Cell name parameter &lt;name&gt; for EXTENT CELL operation not located.  A cell name parameter for an Extent Cell operation is not located in the input layout database.</p> <p>Exception name: EXTENT_CELL</p>
†	<p>EXCEPTION: Cell name parameter &lt;name&gt; for [NOT] INSIDE CELL operation not located.  A cell name parameter for a [NOT] Inside Cell operation is not located in the input layout database.</p> <p>Exception name: INSIDE_CELL</p>
†*	<p>EXCEPTION: Cell &lt;name&gt; is referenced but not defined.  A placement of a cell without a corresponding structure record is encountered.</p> <p>Exception name: MISSING_REFERENCE</p>
†*	<p>EXCEPTION: Another cell record encountered for cell &lt;name&gt;.  A second (etc.) structure record was encountered for a cell.</p> <p>Exception name: DUPLICATE_CELL</p>
†	<p>EXCEPTION: Primary cell &lt;name&gt; selected by wildcard.  The top-level cell was selected by wildcard; that is, its name matched the Layout Primary or Layout Primary2 specification statement parameter (which contained one or more “*” wildcard characters) without the match being literal.</p> <p>Exception name: TOP_CELL_CHOSEN</p>
†*	<p>EXCEPTION: Absolute angle in placement of cell &lt;name&gt; within cell &lt;name&gt; not supported.  A placement has an absolute angle.</p> <p>Exception name: PLACEMENT_ANGLE_ABSOLUTE</p>
†*	<p>EXCEPTION: Absolute magnification in placement of cell &lt;name&gt; within cell &lt;name&gt; not supported.  A placement has an absolute magnification.</p> <p>Exception name: PLACEMENT_MAGNIFICATION_ABSOLUTE</p>

**Table 7-60. Calibre Cell and Placement Exceptions (cont.)**

<b>Comment</b>	<b>Explanation</b>
†*	EXCEPTION: Non-positive magnification in placement of cell <name> not supported. A placement has a non-positive magnification. Exception name: PLACEMENT_MAGNIFICATION_NONPOSITIVE

## Calibre Layout Input Errors

**Table 7-61. Calibre Layout Input Errors**

Explanation
The following fatal errors are reported by all Calibre applications:
<p>ERROR: At least one lef file and exactly one def file are required in LAYOUT PATH when invoked with -fx.  When the Layout System is LEF/DEF, the tool requires at least one lef file and exactly one def file.</p>
<p>ERROR: At least one lef file and exactly one def file are required in LAYOUT PATH2 when invoked with -fx.  When the Layout System is LEF/DEF, the tool requires at least one lef file and exactly one def file.</p>
<p>ERROR: Specified primary cell &lt;name&gt; not located within the input layout database.  The Layout Primary or Layout Primary2 specification statement parameter is not a cell in the input layout database.</p>
<p>ERROR: Specified primary cell &lt;name&gt; is an excluded cell.  The Layout Primary or Layout Primary2 specification statement parameter is an Exclude Cell specification statement parameter.</p>
<p>ERROR: Recursive database reference near cell &lt;name&gt;. A recursive reference is encountered in the input layout database. The structure/symbol name near where the recursive reference was encountered appears in the message.</p>
<p>ERROR: Non-finite or non-positive PLACEMENT magnification &lt;value&gt; in input file &lt;name&gt; at record offset &lt;offset&gt;. The layout database format is OASIS and a PLACEMENT record has an NaN, infinite, or non-positive magnification. The invalid magnification value, file name, and PLACEMENT record offset appears in the message.</p>
<p>ERROR: Non-finite PLACEMENT angle &lt;value&gt; in input file &lt;name&gt; at record offset &lt;offset&gt;. The layout database format is OASIS and a PLACEMENT record has an NaN or infinite rotation. The invalid rotation value, file name, and PLACEMENT record offset appears in the message.</p>
<p>ERROR: Zero COLUMN COUNT value in AREF placement of cell &lt;name&gt; within cell &lt;name&gt;. The layout database format is GDSII and an AREF placement has column count zero.</p>
<p>ERROR: Zero ROW COUNT value in AREF placement of cell &lt;name&gt; within cell &lt;name&gt;. The layout database format is GDSII and an AREF placement has row count zero.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
<p>EXCEPTION: Non-orthogonal rotation (&lt;number&gt;) in AREF placement of cell &lt;name&gt; within cell &lt;name&gt; at location &lt;coordinates&gt;.</p> <p>A GDSII AREF placement has a non-orthogonal rotation. The rotation in degrees, the cell name of the placement, the containing cell name, and the coordinates of the placement are in the message.</p> <p>Exception name: AREF_PLACEMENT</p> <p>Severity is controlled by <a href="#">Layout Input Exception Severity</a>.</p>
<p>EXCEPTION: AREF placement of cell &lt;name&gt; within cell &lt;name&gt;.</p> <p>A GDSII AREF placement has been encountered. The cell name of the placement and the containing cell name are placed into the message. This exception is intended for uses that explicitly prohibit GDSII AREFs.</p> <p>Exception name: AREF_PLACEMENT</p> <p>Severity is controlled by <a href="#">Layout Input Exception Severity</a>.</p>
<p>EXCEPTION: Zero xpitch (ypitch) value in AREF placement of cell &lt;name&gt; within cell &lt;name&gt; at location &lt;point&gt;.</p> <p>A GDSII AREF placement has its x-dimension (y-dimension) &gt; 1 but the spacing in the x-direction (y-direction) is zero. The cell name of the placement, location of the placement, and the containing cell name are placed into the message.</p> <p>Exception Name: ARRAY_PITCH_ZERO</p> <p>Severity is controlled by <a href="#">Layout Input Exception Severity</a>.</p>
<p>EXCEPTION: Zero xspace (yspace) value in placement repetition within cell &lt;name&gt; in input file &lt;name&gt; at location &lt;point&gt; at record offset &lt;offset&gt;.</p> <p>An OASIS placement record has a type 1 or 2 repetition and the x-direction spacing is zero. Or it has a type 1 or 3 repetition and the y-direction spacing is zero. The location of the placement and the containing cell name are placed into the message.</p> <p>Exception Name: ARRAY_PITCH_ZERO</p> <p>Severity is controlled by <a href="#">Layout Input Exception Severity</a>.</p>
<p>ERROR: Non-positive &lt;value&gt; START record in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>The layout database format is OASIS and a START record has an NaN, infinite, or non-positive unit value. The invalid unit value, file name, and START record offset appears in the message.</p>
<p>ERROR: Invalid delta count &lt;count&gt; for type &lt;type&gt; POLYGON point list in input file &lt;file&gt; at file offset &lt;offset&gt;.</p> <p>The delta count for a type 0 or 1 OASIS POLYGON point list is less than two or is odd. The invalid delta count, point list type, file name, and approximate file offset of the delta count appear in the message.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
<p>ERROR: Invalid implicit closure for type &lt;type&gt; POLYGON point list in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>An OASIS POLYGON point list of type 2 has non-orthogonal closure or an OASIS POLYGON point list of type 3 has non-octagonal closure. The point list type, file name, and approximate file offset of the end of the point list.</p>
<p>EXCEPTION: Cell name &lt;name&gt; for LAYOUT POLYGON is not defined.</p> <p>The cell name of a rule file Layout Polygon object is not located in the input layout database.</p> <p>Exception name: LAYOUT_POLYGON</p> <p>Severity is controlled by <a href="#">Layout Input Exception Severity</a>.</p>
<p>ERROR: Invalid magic byte string in input file &lt;name&gt;.</p> <p>The magic string at the beginning of an OASIS file is not “%SEMI-OASIS&lt;CR&gt;&lt;NL&gt;”. The name of the file is placed into the message.</p>
<p>ERROR: Another START record encountered in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>A second OASIS START record is encountered in the same OASIS file. The file name and record offset of the second START record are placed into the message.</p>
<p>ERROR: Record ID &lt;id&gt; encountered outside of CELL record in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>An OASIS PLACEMENT record, geometric record, TEXT record, XELEMENT record, XYRELATIVE record, or XYABSOLUTE record has been encountered outside of a CELL definition. The record ID, file name, and record offset are placed into the message.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
<p>ERROR: Unable to assign modal &lt;string&gt; in input file &lt;name&gt; at record offset &lt;offset&gt;. The value of an OASIS modal variable is undefined. The file name and record offset where the modal value is required to be resolved is placed into the message. &lt;string&gt; identifies the modal value as follows:</p> <ul style="list-style-type: none"> <li>repetition to element</li> <li>point list to POLYGON</li> <li>point list to PATH</li> <li>layer number to geometry</li> <li>layer number to TEXT</li> <li>datatype number to geometry</li> <li>texttype number to TEXT</li> <li>half-width value to PATH</li> <li>start-extension value to PATH</li> <li>end-extension value to PATH</li> <li>width value to RECTANGLE</li> <li>height value to RECTANGLE</li> <li>width value to TRAPEZOID</li> <li>height value to TRAPEZOID</li> <li>type value to CTRAPEZOID</li> <li>width value to CTRAPEZOID</li> <li>height value to CTRAPEZOID</li> <li>radius value to CIRCLE</li> <li>name to PLACEMENT</li> <li>string to TEXT</li> <li>name to PROPERTY</li> <li>value list to PROPERTY</li> </ul>
<p>ERROR: Invalid type for real number in input file &lt;name&gt; at file offset &lt;offset&gt;. A type outside of the range [0,7] for an OASIS real number is present. The file name and offset where the type is encountered is placed into the message.</p>
<p>ERROR: Invalid type for property value in input file &lt;name&gt; at file offset &lt;offset&gt;. A type outside of the range [0,15] for an OASIS property value is present. The file name and offset where the type is encountered is placed into the message.</p>
<p>ERROR: Invalid type for repetition in input file &lt;name&gt; at file offset &lt;offset&gt;. A type outside of the range [0,11] for an OASIS repetition is present. The file name and offset where the type is encountered is placed into the message.</p>
<p>ERROR: Invalid type for point list in input file &lt;name&gt; at file offset &lt;offset&gt;. A type outside of the range [0,5] for an OASIS point list is present. The file name and offset where the type is encountered is placed into the message.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
ERROR: Invalid type for CTRAPEZOID in input file <name> at file offset <offset>. A type outside of the range [0,25] for an OASIS CTRAPEZOID is present. The file name and offset where the type is encountered is placed into the message.
ERROR: No CELLNAME record corresponding to cell name reference number <number> in input file <name>. An OASIS file contains a cell name reference number in a CELL or PLACEMENT record for which there is no corresponding CELLNAME record in the same file. The invalid reference number and file name are placed into the message.
ERROR: No TEXTSTRING record corresponding to text string reference number <number> in input file <name> at record offset <offset>. An OASIS file contains a TEXT record with a text string reference number for which there is no corresponding TEXTSTRING record in the same file. The invalid reference number, file name, and TEXT record offset are placed into the message.
ERROR: No PROPNAME record corresponding to property name reference number <number> in input file <name> at record offset <offset>. An OASIS file contains a PROPERTY record with a property name reference number for which there is no corresponding PROPNAME record in the same file. The invalid reference number, file name, and PROPERTY record offset are placed into the message.
ERROR: No PROPSTRING record corresponding to property string reference number <number> in input file <name> at record offset <offset>. An OASIS file contains a PROPERTY record with a property string reference number for which there is no corresponding PROPSTRING record in the same file. The invalid reference number, file name, and PROPERTY record offset are placed into the message.
ERROR: CELLNAME records 3 and 4 both present in input file <name> at record offset <offset>. An OASIS CELLNAME record with ID 3(4) has been encountered when another CELLNAME record of type 4(3) has been previously encountered in the same file. The file name and CELLNAME record offset are placed into the message.
ERROR: TEXTSTRING records 5 and 6 both present in input file <name> at record offset <offset>. An OASIS TEXTSTRING record with ID 5(6) has been encountered when another TEXTSTRING record of type 6(5) has been previously encountered in the same file. The file name and TEXTSTRING record offset are placed into the message.

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
<p>ERROR: PROPNAMESPACE records 7 and 8 both present in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>An OASIS PROPNAMESPACE record with ID 7(8) has been encountered when another PROPNAMESPACE record of type 8(7) has been previously encountered in the same file. The file name and PROPNAMESPACE record offset are placed into the message.</p>
<p>ERROR: PROPSTRING records 9 and 10 both present in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>An OASIS PROPSTRING record with ID 9(10) has been encountered when another PROPSTRING record of type 10(9) has been previously encountered in the same file. The file name and PROPSTRING record offset are placed into the message.</p>
<p>ERROR: CELLNAME conflict: (name,number) coordinate (&lt;name1&gt;, &lt;number1&gt;) previously resolved as (&lt;name2&gt;, &lt;number2&gt;).</p> <p>Two CELLNAME records in the same OASIS file have the same name but different reference numbers or vice-versa. The conflicting names and numbers are placed into the message.</p>
<p>ERROR: TEXTSTRING conflict: (name,number) coordinate (&lt;name1&gt;, &lt;number1&gt;) previously resolved as (&lt;name2&gt;, &lt;number2&gt;).</p> <p>Two TEXTSTRING records in the same OASIS file have the same name but different reference numbers or vice-versa. The conflicting names and numbers are placed into the message.</p>
<p>ERROR: PROPNAMESPACE conflict: (name,number) coordinate (&lt;name1&gt;, &lt;number1&gt;) previously resolved as (&lt;name2&gt;, &lt;number2&gt;).</p> <p>Two PROPNAMESPACE records in the same OASIS file have the same name but different reference numbers or vice-versa. The conflicting names and numbers are placed into the message.</p>
<p>ERROR: PROPSTRING conflict: (name,number) coordinate (&lt;name1&gt;, &lt;number1&gt;) previously resolved as (&lt;name2&gt;, &lt;number2&gt;).</p> <p>Two PROPSTRING records in the same OASIS file have the same name but different reference numbers or vice-versa. The conflicting names and numbers are placed into the message.</p>
<p>ERROR: CELLNAME record may not post-associate (name, number) coordinate (&lt;name&gt;, &lt;number&gt;).</p> <p>A cell name seen in a CELL or PLACEMENT record and a reference number seen in another CELL or PLACEMENT record may not both appear (that is, become “post-associated”) in a CELLNAME record later in the same OASIS file. The cell name and reference number are placed into the message. If the OASIS file is strict-mode and CALIBRE is run in 64-bit mode, then this exception does not normally occur due to pre-scanning by Calibre.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

<b>Explanation</b>
<p>ERROR: Both H and S bits 1 in info byte for RECTANGLE in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>An OASIS RECTANGLE record has both H and S bits simultaneously set in the info byte. The file name and record offset of the RECTANGLE record are placed into the message.</p>
<p>ERROR: H bit is 1 with type &lt;type&gt; in info byte for CTRAPEZOID in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>An OASIS CTRAPEZOID record of type 16, 17, 18, 19, 22, 23, or 25 has the H bit set in the info byte. The CTRAPEZOID type, file name, and record offset of the CTRAPEZOID record are placed into the message.</p>
<p>ERROR: W bit is 1 with type &lt;type&gt; in info byte for CTRAPEZOID in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>An OASIS CTRAPEZOID record of type 20 or 21 has the W bit set in the info byte. The CTRAPEZOID type, file name, and record offset of the CTRAPEZOID record are placed into the message.</p>
<p>ERROR: INTEGER overflow in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>The value of an OASIS unsigned integer exceeds <math>2^{64} - 1</math> or the absolute value of an OASIS signed integer exceeds <math>2^{63} - 1</math>. The file name and approximate file offset of the integer are placed into the message.</p>
<p>ERROR: REAL with zero denominator in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>An OASIS real number expressed in rational form has a zero denominator. The file name and approximate file offset of the real number are placed into the message.</p>
<p>ERROR: Non b-string of zero length in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>An OASIS a-string or n-string has zero length. The file name and approximate file offset of the string are placed into the message.</p>
<p>ERROR: String of length &lt;length&gt; in input file &lt;name&gt; at file offset &lt;offset&gt; exceeds maximum supported length of 65535.</p> <p>An OASIS string has length exceeding 65535, which is not supported. The length, file name and approximate file offset of the string are placed into the message.</p>
<p>ERROR: Invalid character (&lt;character&gt;) in a-string in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>The OASIS version string (in the START record) or the string in a property value of type 10 has a non-a-string character. The character, file name and approximate file offset of the character are placed into the message.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
<p>ERROR: Component overflow (&lt;name&gt;) in OASIS repetition in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>An OASIS repetition component exceeds the tool's magnitude limits. The name of the component (for example, "x-dimension"), file name, and approximate file offset of the component are placed into the message. This error has the following causes.</p> <p>Define INT_MAX as 2147483647. LONG_MAX is 2147483647 on 32-bit platforms and 18446744073709551614 on 64-bit platforms.</p> <ul style="list-style-type: none"> <li>The x-dimension of a type 1, 2, 4, or 5 repetition exceeds INT_MAX - 2.</li> <li>The y-dimension of a type 1, 3, 6, or 7 repetition exceeds INT_MAX - 2.</li> <li>The x-space of a type 1 or 2 repetition exceeds LONG_MAX.</li> <li>The y-space of a type 1 or 3 repetition exceeds LONG_MAX.</li> <li>The x-space of a type 4 or 5 repetition (after grid adjustment) exceeds INT_MAX.</li> <li>The y-space of a type 6 or 7 repetition (after grid adjustment) exceeds INT_MAX.</li> <li>The n-dimension or m-dimension of a type 8 repetition exceeds INT_MAX - 2.</li> <li>Any displacement component of a type 8 repetition is outside of the interval [-INT_MAX,+INT_MAX].</li> <li>The dimension of a type 9, 10, or 11 repetition exceeds INT_MAX - 2.</li> <li>Any displacement component of a type 9, 10, or 11 repetition (after grid adjustment) is outside of the interval [-INT_MAX,+INT_MAX].</li> </ul>
<p>ERROR: Invalid character (&lt;character&gt;) in n-string in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>An OASIS layer name or the string in a property value of type 12 has a non-n-string character. The character, file name and approximate file offset of the character are placed into the message.</p>
<p>ERROR: Invalid implied a-string reference in property value in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>The string referenced by OASIS property value type 13 has a non-a-string character. The character, file name and approximate file offset of the property value are placed into the message.</p>
<p>ERROR: Invalid implied n-string reference in property value in input file &lt;name&gt; at file offset &lt;offset&gt;.</p> <p>The string referenced by OASIS property value type 15 has a non-n-string character. The character, file name and approximate file offset of the property value are placed into the message.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
<p>ERROR: Invalid property value format for standard property S_GDS_PROPERTY in input file &lt;name&gt; at record offset &lt;offset&gt;. A (standard) OASIS PROPERTY with name S_GDS_PROPERTY is incorrectly formatted. The file name and record offset of the PROPERTY record are placed into the message.</p>
<p>ERROR: Comp-type for CBLOCK record in input file &lt;name&gt; at file offset &lt;number&gt; not supported. The comp-type field in a CBLOCK record has a value other than 0; these are not currently defined in OASIS. The file name and byte offset of the exception is placed into the message.</p>
<p>ERROR: CBLOCK compress format error in input file &lt;name&gt; at file offset &lt;number&gt;. An improper compression format has been located within a CBLOCK record. The file name and byte offset of the exception is placed into the message.</p>
<p>ERROR: Invalid comp-byte-count for CBLOCK record in input file &lt;name&gt; ending at file offset &lt;number&gt;. The number of bytes of compressed data in a CBLOCK record (comp-byte-count) does not match the number when the internal format is actually decompressed. The file name and byte offset terminating the compressed data are placed into the message.</p>
<p>ERROR: Invalid uncomp-byte-count for CBLOCK record in input file &lt;name&gt; ending at file offset &lt;number&gt;. The number of bytes of uncompressed data in a CBLOCK record (uncomp-byte-count) does not match the number when the internal format is actually decompressed. The file name and byte offset terminating the compressed data are placed into the message.</p>
<p>ERROR: START record within CBLOCK record in input file &lt;name&gt; at file offset &lt;number&gt;. An OASIS START record is illegally inside of a CBLOCK record. The file name and byte offset of the exception is placed into the message.</p>
<p>ERROR: END record within CBLOCK record in input file &lt;name&gt; at file offset &lt;number&gt;. An OASIS END record is illegally inside of a CBLOCK record. The file name and byte offset of the exception is placed into the message.</p>
<p>ERROR: CELL record within CBLOCK record in input file &lt;name&gt; at file offset &lt;number&gt;. An OASIS CELL record is illegally inside of a CBLOCK record. The file name and byte offset of the exception is placed into the message.</p>

**Table 7-61. Calibre Layout Input Errors (cont.)**

Explanation
ERROR: CBLOCK record within CBLOCK record in input file <name> at file offset <number>. An OASIS CBLOCK record is illegally inside of a CBLOCK record. The file name and byte offset of the exception is placed into the message.

## Calibre Layout Input Exceptions and Warnings

**Table 7-62. Calibre Layout Input Exceptions and Warnings**

Comment	Explanation
The following warnings are reported by Calibre nmDRC applications:	
	<p>WARNING: Cannot open database input file &lt;filename&gt; for read access. The layout database format is binary or ASCII and the specified input polygon file could not be opened for read access. The run continues as if the file were empty.</p>
The following exceptions are reported by all Calibre applications: Warnings preceded by “†” can have their severities controlled by <a href="#">Layout Input Exception Severity</a> . Warnings preceded by “*” are fatal if <a href="#">Layout Error On Input YES</a> is specified.	
†	<p>EXCEPTION: Layer &lt;number&gt; contains unmapped objects and is the source layer of LAYER MAP &lt;source-layers&gt; DATATYPE &lt;source-datatypes&gt;. A required simple layer number is the source layer of a datatype map but contains objects which themselves are not mapped. The number of the source layer and a representative datatype map are placed into the message. Exception Name: DATATYPE_MAP_SOURCE</p>
†	<p>EXCEPTION: Layer &lt;number&gt; contains unmapped objects and is the target layer of LAYER MAP &lt;source-layers&gt; DATATYPE &lt;source-datatypes&gt;. A required simple layer number is the target layer of a datatype map but contains objects which themselves are not mapped. The number of the target layer and a representative datatype map are placed into the message. Exception Name: DATATYPE_MAP_TARGET</p>
†	<p>EXCEPTION: Unused geometric data is present in the input layout database on layer &lt;number&gt;, datatype &lt;number&gt;. Geometry which is not required in the Calibre run exists in the input layout database. Exception Name: UNUSED_DATA</p>
†	<p>EXCEPTION: A simple layer required in the Calibre run has no geometric data. A simple layer is any layer number, after application of pertinent datatype maps. Exception Name: EMPTY_LAYER</p>
†	<p>EXCEPTION: Geometric data is present in the input layout database on layer &lt;layer&gt;, datatype &lt;datatype&gt; and this layer is <math>\geq</math> to the LAYOUT BUMP2 value. In a dual database application, there is data in the first input layout database whose layer number is greater than or equal to the value of the Layout Bump2 specification statement. These objects are not processed. Exception Name: LAYER_OVER_BUMP</p>

**Table 7-62. Calibre Layout Input Exceptions and Warnings (cont.)**

Comment	Explanation
†	<p>EXCEPTION: Non-closed boundary at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A boundary with more than two vertices and unequal first and last vertices is encountered in a GDSII input layout database. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: POLYGON_NOT_CLOSED</p>
†*	<p>EXCEPTION: Degenerate (vertex count = &lt;number&gt;) polygon at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A polygon has 0 or 1 vertices. Reporting is once per cell with the point reported in cell coordinates if the input layout database format is GDSII, OASIS, LEF/DEF, MILKYWAY, or OPENACCESS; otherwise, reporting is flat with the point reported in world coordinates. The cell name is omitted if the layout database format is Binary or ASCII.</p> <p>Exception Name: POLYGON_DEGENERATE</p>
†*	<p>EXCEPTION: Non-orientable polygon at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A polygon is non-orientable. Reporting is once per cell with the point reported in cell coordinates if the input layout database format is GDSII, OASIS, LEF/DEF, MILKYWAY, or OPENACCESS; otherwise, reporting is flat with the point reported in world coordinates. The cell name is omitted if the layout database format is Binary or ASCII.</p> <p>Exception Name: POLYGON_NONORIENTABLE</p>
†	<p>EXCEPTION: Non-simple polygon at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A polygon or boundary is non-simple but orientable. Reporting is once per cell with the point reported in cell coordinates if the input layout database format is GDSII or OASIS; otherwise, reporting is flat with the point reported in world coordinates. The cell name is omitted if the layout database format is Binary or ASCII.</p> <p>Exception Name: POLYGON_NONSIMPLE</p>
†	<p>EXCEPTION: Acute polygon at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A GDSII boundary or OASIS polygon has an acute angle. Reporting is once per cell with the point reported in cell coordinates.</p> <p>The default severity is 0. Severity may be explicitly set with the specification statement Layout Input Exception Severity. Layout Error On Input does not affect this exception.</p> <p>Exception Name: POLYGON_ACUTE</p>

**Table 7-62. Calibre Layout Input Exceptions and Warnings (cont.)**

Comment	Explanation
†	<p>EXCEPTION: Zero-area polygon at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A GDSII boundary or OASIS polygon has zero area. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Severity may be explicitly set with the specification statement Layout Input Exception Severity. Layout Error On Input does not affect this exception.</p> <p>Exception Name: POLYGON_AREA_ZERO</p>
†	<p>EXCEPTION: Two-point boundary at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A two-point boundary is encountered in a GDSII input layout database. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: POLYGON_IS_RECTANGLE</p>
†	<p>EXCEPTION: (Rectangle   2-point boundary   box) of zero length or width at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A rectangle has zero length or width. Reporting is once per cell with the point reported in cell coordinates. Note that 2-point boundaries in input layout databases are interpreted by Calibre as orthogonal rectangles.</p> <p>Exception Name: RECTANGLE_SIDE_ZERO</p>
†*	<p>EXCEPTION: Degenerate (vertex count = &lt;number&gt;) path at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path has 0 or 1 vertices. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_DEGENERATE</p>
†*	<p>EXCEPTION: Non-orientable path at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path is non-orientable. A non-orientable path is a path that produces a non-orientable polygon when expanded. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_NONORIENTABLE</p>
†*	<p>EXCEPTION: Large vertex path (&gt;1024 vertices) at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path has more than 1024 vertices. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_BIG</p>
†	<p>EXCEPTION: Non-simple path at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path is non-simple but orientable. A non-simple path is one whose expansion is a non-simple polygon. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_NONSIMPLE</p>

**Table 7-62. Calibre Layout Input Exceptions and Warnings (cont.)**

Comment	Explanation
†	<p>EXCEPTION: Path of odd width (&lt;number&gt; dbu) at location &lt;coordinates&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A GDSII path has a width that is not divisible by 2. Reporting is hierarchical and the width of the path is placed into the message along with the location, cell name, layer, and datatype.</p> <p>Exception Name: PATH_WIDTH_ODD</p>
†	<p>EXCEPTION: Path of zero width at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path has zero width and has been ignored. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_WIDTH_ZERO</p>
†*	<p>EXCEPTION: Spike in path at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path has a spike (the path doubles back upon itself). Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_SPIKE</p>
†*	<p>EXCEPTION: Path of absolute width at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; not yet supported.</p> <p>A layout path with absolute width is encountered. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_WIDTH_ABSOLUTE</p>
†*	<p>EXCEPTION: Problem extending (type 4   variable end) path at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt; by &lt;number&gt;.</p> <p>The system failed to expand a type-4 GDSII or variable-end OASIS path.</p> <p>Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_EXTENSION</p>
†	<p>EXCEPTION: Path endsegment length &lt;number&gt; at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt; is less than 1/2 of path width &lt;number&gt;.</p> <p>The endsegment length of a (multiple-segment) type-0 GDSII path or flush-ended OASIS path is less than 1/2 of the path width and expansion of the path by the given width would result in a notch at the short end. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_ENDSEGMENT_SHORT</p>
†	<p>EXCEPTION: Angled path at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path whose centerline contains one or more angled segments has been encountered. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_ANGLLED</p>

**Table 7-62. Calibre Layout Input Exceptions and Warnings (cont.)**

Comment	Explanation
†	<p>EXCEPTION: Path with acute angle at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path whose centerline contains one or more acute angles has been encountered. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_ACUTE</p>
†	<p>EXCEPTION: Path with coincident endpoints at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A layout path whose centerline endpoints coincide has been encountered. Reported once.</p> <p>Exception Name: PATH_COINCIDENT</p>
†	<p>EXCEPTION: Path of type 1 at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>A type-1 (circular endcap) GDSII path has been encountered. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: PATH_CIRCULAR</p>
†*	<p>EXCEPTION: Zero-vertex text object &lt;name&gt; in cell &lt;name&gt; on layer &lt;number&gt;.</p> <p>A text object has more than one vertex. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: TEXT_DEGENERATE</p>
†*	<p>EXCEPTION: Multi-vertex text object &lt;name&gt; at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;number&gt;.</p> <p>A text object has more than one vertex. Reporting is once per cell with the point reported in cell coordinates.</p> <p>Exception Name: TEXT_BIG</p>
†	<p>EXCEPTION: Cell name &lt;name&gt; for LAYOUT TEXT object &lt;name&gt; is not defined.</p> <p>The cell name of the rule file LAYOUT TEXT object having the given name is not located in the input layout database.</p> <p>Exception Name: LAYOUT_TEXT</p>
†*	<p>EXCEPTION: Invalid call (rotation is (0,0)) of symbol &lt;number&gt; in cell &lt;name&gt;.</p> <p>A symbol call with rotation (0,0) is encountered.</p> <p>Exception Name: CALL_ROTATION</p>
†	<p>EXCEPTION: Circle at location &lt;point&gt; in cell &lt;name&gt; on layer &lt;layer&gt; datatype &lt;datatype&gt;.</p> <p>An OASIS circle has been encountered. Reporting is once per cell with the (center) point reported in cell coordinates.</p> <p>Exception Name: CIRCLE</p>

**Table 7-62. Calibre Layout Input Exceptions and Warnings (cont.)**

Comment	Explanation
†	EXCEPTION: Circle of radius zero at location <point> in cell <name> on layer <layer> datatype <datatype>. An OASIS circle has zero diameter. Reporting is once per cell with the point reported in cell coordinates. Exception Name: CIRCLE_RADIUS_ZERO
†	EXCEPTION: Unsupported XGEOMETRY at location <point> in cell <name> on layer <layer> datatype <datatype>. An OASIS XGEOMETRY record (not supported) has been encountered. Reporting is once per cell with the point reported in cell coordinates. Exception Name: XGEOMETRY_UNSUPPORTED
†	EXCEPTION: Degenerate TRAPEZOID at location <point> in cell <name> on layer <layer> datatype <datatype>. An OASIS TRAPEZOID record is degenerate (that is, cannot be formed into a polygon as specified). Reporting is once per cell with the point reported in cell coordinates. Exception Name: TRAPEZOID_DEGENERATE
†	EXCEPTION: Degenerate CTRAPEZOID at location <point> in cell <name> on layer <layer>. An OASIS CTRAPEZOID record is degenerate that is, cannot be formed into a polygon as specified). Reporting is once per cell with the point reported in cell coordinates. Exception Name: CTRAPEZOID_DEGENERATE
†	EXCEPTION: TRAPEZOID of zero area at location <point> in cell <name> on layer <layer> datatype <datatype>. An OASIS TRAPEZOID has zero area. Reporting is once per cell with the point reported in cell coordinates. Exception Name: TRAPEZOID_AREA_ZERO
†	EXCEPTION: CTRAPEZOID of zero area at location <point> in cell <name> on layer <layer> datatype <datatype>. An OASIS CTRAPEZOID has zero area. Reporting is once per cell with the point reported in cell coordinates. Exception Name: CTRAPEZOID_AREA_ZERO
†	EXCEPTION: BOX record at location <point> in cell <name> on layer <layer> datatype <datatype>. A GDSII BOX record has been encountered. Reporting is once per cell with the point reported in cell coordinates. Exception Name: BOX_RECORD
†	EXCEPTION: NODE record at location <point> in cell <name> on layer <layer> datatype <datatype>. A GDSII NODE record has been encountered. Reporting is once per cell with the point reported in cell coordinates. Exception Name: NODE_RECORD

**Table 7-62. Calibre Layout Input Exceptions and Warnings (cont.)**

Comment	Explanation
†	<p>EXCEPTION: XNAME record in input file &lt;name&gt; at record offset &lt;number&gt; not supported.</p> <p>An OASIS XNAME record (unsupported) has been encountered. The file name and record offset of the XNAME record is placed into the message.</p> <p>Exception Name: XNAME_UNSUPPORTED</p>
†	<p>EXCEPTION: XELEMENT record in input file &lt;name&gt; at record offset &lt;number&gt; not supported.</p> <p>An OASIS XELEMENT record (unsupported) has been encountered. The file name and record offset of the XELEMENT record is placed into the message.</p> <p>Exception Name: XELEMENT_UNSUPPORTED</p>
†	<p>EXCEPTION: Non n-string for CELLNAME record in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>The string in a CELLNAME, CELL, or PLACEMENT record is not an OASIS n-string. The file name and record offset are placed into the message.</p> <p>Exception Name: CELLNAME_NSTRING</p>
†	<p>EXCEPTION: Non a-string for TEXTSTRING record in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>The string in a TEXTSTRING or TEXT record is not an OASIS a-string. The file name and record offset are placed into the message. If available, the cell name, text object location, and text object layer are also inserted into the message.</p> <p>Exception Name: TEXTSTRING_ASTRING</p>
†	<p>EXCEPTION: Non n-string for PROPNAMERecord in input file &lt;name&gt; at record offset &lt;offset&gt;.</p> <p>The string in a PROPNAMERecord or PROPERTY record is not an OASIS n-string. The file name and record offset are placed into the message.</p> <p>Exception Name: PROPNAMERecord_NSTRING</p>

## Calibre GDSII and OASIS Output Warnings

**Table 7-63. Calibre GDSII and OASIS Output Warnings**

Explanation
The following warnings are reported by Calibre nmDRC applications:  WARNING: RuleCheck output to GDSII results database <name> with maximum polygon vertex count of <number> (more than 199). A nmDRC rule check will output to a GDSII type nmDRC results database with a maximum polygon vertex count greater than 199 (the legal GDSII maximum). This warning is printed for each nmDRC results database so generated and the file name of the database and the actual maximum vertex value is placed into the message.
WARNING: RuleCheck output to (GDSII or OASIS) results database <name> with maximum result count of <number> (less than ALL). A nmDRC rule check will output to a GDSII or OASIS type nmDRC results database with a maximum result count less than ALL. This warning is printed for each results database so generated and the file name of the database and the actual maximum result value is placed into the message. The intent is to warn of inadvertent use of the default value 1000 when writing mask data versus DRC errors. This warning is printed in the CALIBRE DRC results database initialization module and setting the environment variable CALIBRE_EXIT_MAXIMUM_RESULTS to a non-NULL value will cause the application to immediately abort after all such warnings are printed.
WARNING: RuleCheck <name> output to (GDSII or OASIS) results database <name> with default layer 0, datatype 0. A nmDRC rule check outputs to a GDSII or OASIS type nmDRC results database with a default layer number (namely 0). This may occur because a layer number is not specified in the drc check map specification for the rule check or because no drc check map is specified for the rule check. This warning is printed for each nmDRC rule check which outputs a default layer number to a GDSII or OASIS type nmDRC results database and the names of the rule check and the results database are placed into the message. This warning is printed in the CALIBRE DRC results database initialization module and setting the environment variable CALIBRE_EXIT_LAYER_DEFAULT to a non-NULL value will cause the application to immediately abort after all such warnings are printed.
WARNING: Structure names longer than 32 characters have been written to GDSII nmDRC results database <name>. One or more structure names written to an output GDSII-type nmDRC results database has a length exceeding 32 characters. This length limit is a GDSII anachronism and is rarely enforced by current stream readers; hence, this warning is simply meant to be informative. The file name of the affected nmDRC results database appears in the message.

**Table 7-63. Calibre GDSII and OASIS Output Warnings (cont.)**

Explanation
The following are reported by Calibre nmDRC-H:  WARNING: DRC CHECK MAP AREF cell name <name> duplicates layout database cell name—AREF output suppressed. An AREF cell name in a DRC Check Map specification statement coincides with a placed cell name in the input layout database. AREF output is not performed for that particular AREF specification. The name of the cell appears in the message.
WARNING: DRC MAXIMUM CELL NAME LENGTH truncation of output cell name <name> to <number> characters is not possible. A structure name written to an output GDSII- or OASIS-type nmDRC results database has a length exceeding that specified in the DRC Maximum Cell Name Length specification statement. This is due to the fact that the name could not be truncated to any positive length and avoid duplication with any other output cell name. The name of the cell and the DRC Maximum Cell Name Length parameter appears in the message.

## Calibre Output Warnings

**Table 7-64. Calibre Output Warnings**

Explanation
<p>The following warning is reported by Calibre applications:</p> <p><b>WARNING: TEXT PRINT MAXIMUM limit of &lt;number&gt; exceeded.</b> &lt;number&gt; objects not printed.</p> <p>The number of text objects printed in the transcript would exceed the value specified in the Text Print Maximum specification statement. Only that number are printed and the number not printed appears in the message.</p>

## Cell Name and Location Warnings

**Table 7-65. Cell Name/Location Warnings**

Explanation
<p>The following warnings are reported by all Calibre applications:</p> <p><b>WARNING: Cell name parameter &lt;name&gt; for LAYOUT WINDOW/WINDEL CELL specification statement not located.</b> A cell name parameter for a Layout Window Cell or Layout Windel Cell specification statement is not located and has been ignored</p>
<p>The following warnings are reported by all hierarchical Calibre applications:</p> <p><b>WARNING: EXPAND CELL &lt;name&gt; not located or not allowed.</b> A cell in an Expand Cell specification statement is not located in the input database or is the top-level cell.</p>
<p><b>WARNING: FLATTEN CELL &lt;name&gt; not located or not allowed.</b> A cell in a Flatten Cell specification statement is not located in the input database or is the top-level cell.</p>
<p><b>WARNING: FLATTEN INSIDE CELL &lt;name&gt; not located or not allowed.</b> A cell in a Flatten Inside Cell specification statement is not located in the input database or is the top-level cell.</p>
<p><b>WARNING: PUSH CELL &lt;name&gt; not located or not allowed.</b> A cell in a Push Cell specification statement is not located in the input database or is the top-level cell.</p>
<p><b>WARNING: HCELL &lt;name&gt; not located or not allowed.</b> A cell in an Hcell specification statement is not located in the input database, or is the top-level cell, or has no placements in the cell.</p>
<p><b>WARNING: LAYOUT BASE CELL &lt;name&gt; not located or not allowed.</b> A cell in a layout base cell specification statement is not located in the input database or is the top-level cell.</p>

**Table 7-65. Cell Name/Location Warnings (cont.)**

Explanation
The following is reported by ICVerify:  WARNING: EXCLUDED CELL <name> would not be processed. Reported for every excluded cell name parameter to CHEck DRC which would not have been processed anyway. The most common cause of this warning is a typographic error.

## Connectivity Extraction and Stamp Warnings

**Table 7-66. Connectivity Extraction and Stamp Warnings**

Explanation
The following warnings are reported by all Calibre nmDRC applications and by the ICrules CHEck DRc command:
<p><b>WARNING:</b> There is no data for layout net name &lt;name&gt;. The net name parameter of a Net or Not Net operation required by the check set is not the name of a net formed by the connectivity extractor. (Execution will continue with an empty layer generated for Net and the input layer generated for Not Net).</p>
<p><b>WARNING:</b> Missing connections STAMPing layer &lt;layer1&gt; by layer &lt;layer2&gt;. A Stamp &lt;layer1&gt; By &lt;layer2&gt; operation has been performed in a nmDRC run and some data on &lt;layer1&gt; was not output by the operation due to a missing connection on &lt;layer2&gt;. This message is only issued once per Stamp operation. The names of the layer parameters to the operation are placed in the message.</p>
<p><b>WARNING:</b> Conflicting connections STAMPing layer &lt;layer1&gt; by layer &lt;layer2&gt;. During a Stamp &lt;layer1&gt; By &lt;layer2&gt; operation, some data on &lt;layer1&gt; was not output by the operation due to a multiple connection on &lt;layer2&gt;. This message is only issued once per Stamp operation. The names of the layer parameters to the operation are placed in the message.</p>
<p><b>WARNING:</b> Short circuit [in cell &lt;name&gt;]. Label &lt;name&gt; at location &lt;point&gt; used. Label &lt;name&gt; at location &lt;point&gt; ignored. Connectivity extraction for nmDRC detected a redundant net name. This warning is suppressed in an area-based check in ICrules. Reporting is flat with the points reported in world coordinates. The cell name is reported in Calibre nmDRC-H (with points reported in cell coordinates) if the DRC Cell Text YES statement was specified.</p>
<p><b>WARNING:</b> Open circuit [in cell &lt;name&gt;]. Label &lt;name&gt; used at location &lt;point&gt; and ignored at location &lt;point&gt;. Connectivity extraction for nmDRC detected an attempt to assign the same name to two different nets. This warning is suppressed in an area-based check in ICrules. Reporting is flat with the points reported in world coordinates. The cell name is reported in Calibre nmDRC-H (with points reported in cell coordinates) if the DRC Cell Text YES statement was specified.</p>
<p><b>WARNING:</b> Label &lt;name&gt; at location &lt;point&gt; [in cell &lt;name&gt;] intersects multiple nets; one net chosen arbitrarily. Connectivity extraction for nmDRC detected a label intersecting and attached to two different nets. This warning is suppressed in an area-based check in ICrules. Reporting is flat with the points reported in world coordinates. The cell name is reported in Calibre nmDRC-H (with points reported in cell coordinates) if the DRC Cell Text YES statement was specified.</p>

**Table 7-66. Connectivity Extraction and Stamp Warnings (cont.)**

Explanation
<p>WARNING: Label &lt;name&gt; at location &lt;point&gt; [in cell &lt;name&gt;] is unattached and has been ignored.</p> <p>Connectivity extraction for nmDRC detected an unassigned label. This warning is suppressed in an area-based check in ICrules. Reporting is flat with the points reported in world coordinates. The cell name is reported in Calibre nmDRC-H (with points reported in cell coordinates) if the DRC Cell Text YES statement was specified. The total number of these warnings in the transcript are user-controllable with the specification statement DRC Maximum Unattached Label Warnings &lt;value&gt;, where &lt;value&gt; is a non-negative integer.</p>
<p>WARNING: Maximum unattached label warning count of &lt;number&gt; exceeded. &lt;number&gt; unattached label warnings suppressed.</p> <p>The rule file is limiting the number of unattached label warnings with the specification statement DRC Maximum Unattached Label Warnings. The limit has been exceeded and warnings beyond the specified limit have been suppressed. The number of unreported unattached labels is placed into the message.</p>
<p>WARNING: Virtually connected label &lt;name&gt; at location &lt;point&gt; and location &lt;point&gt;.</p> <p>Reported whenever connectivity extraction for nmDRC forms a virtual connection. This warning is suppressed in an area-based check in ICrules. Reporting is flat with the points reported in the world coordinates. This warning is only reported if the Virtual Connect Report YES statement was specified</p>
<p>WARNING: Sconnect conflict. Multiple stamping occurred.</p> <p>An Sconnect operation, executed during connectivity extraction for nmDRC, encountered a conflicting unidirectional connection and chose a single node.</p>
<p>The following warnings are reported by the ICrules CHEck DRc command:</p>
<p>WARNING: Port &lt;name&gt; at location &lt;point&gt; is unattached and has been ignored.</p> <p>Connectivity extraction for nmDRC detected a port which cannot be used to establish connectivity. This warning is suppressed in an area-based check. Reporting is flat with the point reported in world coordinates.</p>
<p>WARNING: Feedthrough at location &lt;point&gt; is unattached and has been ignored.</p> <p>Connectivity extraction for nmDRC detected an explicit connect which cannot be used to establish connectivity. This warning is suppressed in an area-based check. Reporting is flat with the point reported in world coordinates.</p>

## DFM Warnings

**Table 7-67. DFM Warnings**

Explanation
WARNING: Call for value of the displayed property on the displayed layer will return 0.0 — property not found.
WARNING: The type of the displayed property on the displayed layer is not “number”.
WARNING: The displayed property on the displayed layer is not a polygon property.
WARNING: PROPERTY() calls on the displayed layer will return 0.0 regardless of the property name.
WARNING: Call for value of the displayed property on the displayed layer will return 0.0 — no properties on that layer.
WARNING: Type of the displayed property on the displayed layer is not “vector of number”.
WARNING: Call for value of the displayed property on the displayed layer will return empty vector with tuple-size 1 — property not found.
WARNING: The displayed property on the displayed layer is referenced with more than one type of value.
WARNING: The type of the displayed property on the displayed layer is not “netID”.
WARNING: The type of the displayed property on the displayed layer is not “vector of netID”.
WARNING: Call for value of the displayed property on the displayed layer will return an invalid NETID value — no properties on that layer.
WARNING: Call for value of the displayed property on the displayed layer will return an invalid NETID value — property not found.
WARNING: The type of the displayed property on the displayed layer is not “vector of uniqID”.
WARNING: The displayed property was not found on the displayed layer.
WARNING: The type of the displayed property on the displayed layer is not “string”.
WARNING: DFM Connectivity Redundant is not supported in flat Calibre applications; output layer will be empty.
WARNING: PRINT POLYGONS option for PATHCHK is not supported in DFM applications.

**Table 7-67. DFM Warnings (cont.)**

Explanation
WARNING: PRINT NETS option for PATHCHK is not supported in DFM applications.
WARNING: RDB output for DFM Analyze will be saved to the DFM database instead of the RDB file.
WARNING: RDB and RDB ONLY options for DFM Measure are not supported in DFM applications.
WARNING: RDB and RDB ONLY options for DFM Transition are not supported in DFM applications.
WARNING: Input layers for DFM RDB will be saved to the DFM database instead of the RDB file.

## ICrules Input Warnings

**Table 7-68. ICrules Input Warnings**

Explanation
The following warnings are reported by the ICrules CHEck DRc command:
WARNING: <number>-vertex polygon with handle <handle> on layer <layer> in cell <name> has been ignored. CHEck DRc detected a degenerate object (a polygon with less than three vertices) in the Pyxis Layout database. The object is ignored.
WARNING: Zero-width path with handle <handle> on layer <layer> in cell <name> has been ignored. CHEck DRc detected a zero-width path in the Pyxis Layout database. The path is ignored.

## ICrules Notes

**Table 7-69. ICrules Notes**

Explanation
The following notes are printed by the ICrules CHEck DRc command.
NOTE: Layer <layer> read. Original Geometry count: <number>. Shown after an original layer has been read by CHEck DRc from the Pyxis Layout database.
NOTE: PINS from <layer> read. PIN count: <number>. Shown after a Pins operation has been executed by CHEck DRc.
NOTE: PORTS from <layer> read. PORT count: <number>. Shown after a Ports operation has been executed by CHEck DRc.
NOTE: TOPEX from <layer> read. TOPEX count: <number>. Shown after a Topex operation has been executed by CHEck DRc.
NOTE: DRC RuleCheck <name> completed; Result count: <number>. Shown after a rule check statement has been executed by CHEck DRc. <number> is the number of objects placed into the nmDRC results database under the rule check statement with the name <name>.
NOTE: DRC RuleCheck <name> concluded due to interrupt; Result count: <number>. Shown after a rule check statement has been executed by CHEck DRc, but concluded early due to an interrupt. <number> is the number of objects placed into the nmDRC results database under the rule check statement with the name <name>.

**Table 7-69. ICrules Notes (cont.)**

Explanation
<p>NOTE: DRC completed. Total RuleChecks: &lt;number&gt;; Total Results: &lt;number&gt;; Total Original Geometries: &lt;number&gt;; CPU Time: &lt;time&gt;, REAL Time: &lt;time&gt;.</p> <p>Shown when CHEck DRc completes. The statistics include the total number of rule check statements executed, the total number of nmDRC results, and the total number of original shapes input to the CHEck DRc run.</p>

## LVS Connectivity Extraction Messages

**Table 7-70. LVS Connectivity Extraction Messages**

Explanation
The following errors and warnings are reported by Calibre applications:
<p><b>ERROR:</b> The unconditional must-connect condition between the following instance pins has not been satisfied.</p> <p>Pins of an instance that belong to a single unconditional must-connect group in the instance's template are not connected in the cell that contains the instance and cannot be connected at any higher level. The pin names are reported in the form: <i>&lt;instance_name&gt;. &lt;pin_name&gt;</i>.</p>
<p><b>ERROR:</b> The conditional must-connect condition between the following instance pins has not been satisfied.</p> <p>Pins of an instance that belong to a single conditional must-connect group in the instance's template are not connected in the cell that contains the instance, and cannot be connected at any higher level. This error is reported only if a connection was made to one of the pins. The pin names are reported in the form: <i>&lt;instance_name&gt;. &lt;pin_name&gt;</i>.</p>
<p><b>ERROR:</b> A must-connect condition exists on instance pins but is not defined for the corresponding ports.</p> <p>Pins of an instance that belong to a single must-connect group in the instance's template and that are not connected in the containing cell, are connected instead to ports of the cell, but the ports do not belong to a single must-connect group in the cell. The instance pin names in the form <i>&lt;instance_name&gt;. &lt;pin_name&gt;</i> and the corresponding port names are reported.</p>
<p><b>ERROR:</b> Can not determine MASK SVDB directory from rule file <i>&lt;name&gt;</i> for extracted netlist output. Specify MASK SVDB directory or use “-spice <i>&lt;filename&gt;</i>” command line option.</p> <p>The Layout System is geometric and calibre -lvs -hier has been specified on the command line. In order to extract the layout SPICE netlist, either the rule file must have a Mask SVDB Directory statement in it, or the -spice command line option must be used.</p>
<p><b>Error:</b> The following products could not be licensed sufficiently:</p> <p><b>Error:</b> - LVS Device DFM Properties</p> <p>The DFM Property operation is used to calculate Device properties. This requires a Calibre ADP license.</p>
<p><b>Error:</b> The following products could not be licensed sufficiently:</p> <p><b>Error:</b> - LVS Device TVF</p> <p>There is a TVF FUNCTION block that requires a Calibre ADP license.</p>
<p><b>WARNING:</b> Port contains unconnected shapes. It can contribute false feedthroughs on instances of the cell.</p> <p>Issued for a port with multiple shapes or paths that are not physically connected inside the cell. The port name is reported.</p>

**Table 7-70. LVS Connectivity Extraction Messages (cont.)**

Explanation
<p><b>WARNING:</b> Direct connection between different ports.</p> <p>A direct connection was made between distinct ports of the cell. The port names are reported. You can disable this warning by including the LVS Report Option P specification statement in your rule file.</p>
<p><b>WARNING:</b> Short circuit — Different names on one net.</p> <p>This warning is issued when several different names are found on a single net. One name is chosen and the other names are ignored. Conflicts are resolved in favor of, in order of precedence: (1) power names, if specified, in rule file order; (2) ground names, if specified, in rule file order; and (3) the last name in alphabetical order. Power and ground names are specified with optional LVS Power Name and LVS Ground Name statements in the rule file. All names found on the net are reported, along with their locations and layers. The name actually assigned to the net is indicated. In mask mode extraction, the net ID is also reported.</p>
<p><b>WARNING:</b> Open circuit — Same name on different nets.</p> <p>This warning is issued when an attempt is made to assign the same name to two or more different nets. One of the nets is arbitrarily chosen and the other nets are left unnamed. The name is reported, along with all locations and layers on which it was found. In mask mode extraction, the report also includes the net IDs on which the name was found, and the net ID to which the name was actually assigned.</p>
<p><b>WARNING:</b> Ambiguous label attachment — One label intersecting different nets.</p> <p>This warning is issued when a label intersects the regions of two or more different nets and an attempt is made to assign the label to those nets by means of Attach or Label Order operations in the rule file. One of the nets is arbitrarily chosen. The label name and its location and layer are reported. In mask mode extraction, the report also includes the IDs of all nets involved in the conflict, and the ID of the net to which the label was actually assigned.</p>
<p><b>WARNING:</b> Unattached label.</p> <p>This warning is issued for a label that cannot be assigned to any net. The label name, location, and layer are reported.</p>
<p><b>WARNING:</b> Unattached feedthrough or overflow pads; feedthrough or overflow ignored.</p> <p>This warning is issued in Pyxis Layout when a feedthrough (instance pin with multiple shapes) or overflow cannot be used to establish connectivity. This happens when an Attach operation in the rule file cannot be applied to the feedthrough or overflow because the target layer of the Attach is not present at the location of the feedthrough or overflow. The location and layer are reported.</p>

**Table 7-70. LVS Connectivity Extraction Messages (cont.)**

Explanation
WARNING: Unattached must-connect pad; must-connect condition ignored. This warning is issued when a must-connect condition on an instance pin cannot be used. This happens when an Attach operation in the rule file cannot be applied to the must-connect because the target layer of the Attach is not present at the location of the must-connect. The must-connect number, pin name, layer and location are reported.
WARNING: Unattached port pads; port ignored. This warning is issued when a port cannot be associated with any net. This occurs when the port layer does not appear in Connect, Attach, or Label Order specification statements, or when there is no geometry at the port location that the port can be attached to. The port name, layer and location are reported.
WARNING: Hcells “ <i>cellname</i> ” and “ <i>cellname</i> ” have incompatible port swap sets - correspondence ignored. This warning is issued when one or more pairs of component types that are specified as corresponding in the hcell list have different pin swappability. The hcell correspondence is ignored.
WARNING: Hcells “ <i>cellname</i> ” and “ <i>cellname</i> ” are of different kinds (“LVS box cell or empty subcircuit”) and (“Subcircuit”) - correspondence ignored. For example, the hcell list contains the pair “AAA BBB” but AAA is a cell and BBB is a primitive device; or, AAA is defined to be a CMOS N transistor and BBB is defined to be a CMOS P transistor. This warning is also given if one hcell of a corresponding pair is an empty subckt or an <a href="#">LVS Box</a> cell. Detailed information appears in the transcript. The correspondence in such cases is ignored.
WARNING: Hcells “ <i>cellname</i> ” and “ <i>cellname</i> ” have different property rules - correspondence ignored. This warning is issued when one or more pairs of component types that are specified as corresponding in the hcell list have different Trace Property rules specified. The hcell correspondence is ignored.
WARNING: ENCLOSURE_PERPENDICULAR/ENCLOSURE_PARALLEL may require excessive computations for this design -- consider using DFM PROPERTY This warning is issued during connectivity extraction if Device statement property computations use the ENCLOSURE_PERPENDICULAR or ENCLOSURE_PARALLEL functions and they will require excess runtime to perform their computations. In this case, the DFM Property operation can be useful in reducing runtime. See “ <a href="#">instance</a> ” on page 1817.

**Table 7-70. LVS Connectivity Extraction Messages (cont.)**

Explanation
<p>NOTE: Virtually connected...</p> <p>This note indicates a name-based virtual connection. For example, from the Virtual Connect Name or Virtual Connect Colon specification statements. For each pair of labels that cause a virtual connection, the label name and both label locations are indicated. This reporting is enabled with the LVS Report Option V specification statement.</p>

## LVS Comparison Messages

**Table 7-71. LVS Comparison Messages**

Explanation
The following errors and warnings are reported by Calibre applications:
<p>ERROR: Duplicate cell name “&lt;name&gt;” in cell lists “&lt;list&gt;” and “&lt;list&gt;” after wildcard substitution.</p> <p>Wildcards are used in an LVS Cell List statement, resulting in a cell being matched by two different lists. The conflict must be resolved.</p>
<p>ERROR: Parameter “&lt;name&gt;” ambiguously renamed in file “&lt;netlist&gt;” at line &lt;line&gt;</p> <p>LVS Spice Rename Parameter is specified and a parameter name is ambiguously renamed.</p>
<p>ERROR: Supply error detected. ABORT ON SUPPLY ERROR is specified - aborting.</p> <p>LVS Abort On Supply Error is specified and a supply error has been detected. Circuit comparison is aborted.</p>
<p>ERROR: Property &lt;prop&gt; was declared string in TRACE PROPERTY user program, but (layout) input contains numeric data.</p> <p>A user-defined Trace Property program declares a property as a string, but the property is numeric in the layout.</p>
<p>WARNING: Cells “&lt;name&gt;” ...expanded due to seed promotion - correspondence ignored.</p> <p>LVS Expand Seed Promotions YES is specified and cells have been expanded accordingly.</p>
<p>WARNING: Duplicate subckt definition “&lt;subcircuit&gt;” at line &lt;N&gt; in file “&lt;file&gt;”</p> <p>LVS Spice Allow Duplicate Subcircuit Names YES is specified and a duplicate name is found while parsing a netlist.</p>
<p>WARNING: Duplicate subckt name “&lt;name&gt;” at line &lt;N&gt; in file “&lt;file&gt;”</p> <p>LVS Spice Option S is specified and a duplicate subcircuit name is detected.</p>
<p>WARNING: Floating pins in file “&lt;file&gt;” at line &lt;N&gt; (&lt;N&gt; pins)</p> <p>LVS Spice Option F is specified and floating pins are detected.</p>
<p>WARNING: lvs cpoint name &lt;name&gt; not found in &lt;design&gt;.</p> <p>LVS Cpoint is specified but a name cannot be found in one of the designs.</p>
<p>WARNING: LVS property resolution maximum exceeded.</p> <p>The LVS Property Resolution Maximum value has been exceeded. Default is 32. Not all circuit ambiguities were processed by ambiguity resolution. False property errors may result.</p>

**Table 7-71. LVS Comparison Messages (cont.)**

Explanation
WARNING: Undefined parameter “<param>” ignored in file “<file>” at line <N> An undefined parameter has been found in a SPICE netlist. Possibly using LVS Spice Allow Inline Parameters YES can resolve the problem if it is an inline parameter. See “ <a href="#">Element Statements</a> ” in the <i>Calibre Verification User’s Manual</i> for a description of correct SPICE syntax for built-in elements.

## LVS Stamp Error Messages

**Table 7-72. LVS Stamp Error Messages**

Explanation
The following errors and warnings are reported by Calibre applications:
ERROR: Missing connections STAMPing layer x by layer y. A list of locations where layer x cannot be stamped by layer y is provided. Each location is a vertex of a polygon on layer x that is not overlapped by any polygon on layer y.
ERROR: Conflicting connections STAMPing layer x by layer y. A list of locations where the node reference assignment is ambiguous, that is, where a polygon on layer x is overlapped by two or more polygons on layer y belonging to different electrical nets. For each location, the net IDs and net names of two of the conflicting nets are provided. In addition, one vertex of the layer x polygon is provided.

## LVS Netlist Compilation Errors and Warnings

When a SPICE netlist is used as input, a transcript of any errors and warnings found during compilation are listed near the top of the LVS Report file. For descriptive information about these messages, refer to the “[LVS Results](#)” chapter in the *Calibre Verification User’s Manual*. Errors cause LVS to abort but warnings do not.

## PERC TVF Errors

During PERC Load initialization or rule checking, PERC executes Tcl procs. PERC captures runtime problems as Tcl errors and subsequently reports them in the TVF FUNCTION ERROR section of the report. The following is a list of the errors:

**Table 7-73. PERC TVF Errors**

Error Message and Description
ERROR: cannot call rule check commands during initialization.  This error is issued if any rule check command is called in initialization procedures of PERC LOAD statements.
ERROR: cannot call initialization commands outside an init proc.  This error is issued if any initialization command is called in rule checks.
ERROR: cannot call perc::report_base_result outside of rule checking commands.  This error is issued if perc::report_base_result is called at the time when none of the rule checking commands is in progress.
ERROR: cannot descend from the non-hcell instance: <name>.  This error is issued if the argument to perc::descend is not an instance iterator pointing to a cell instance, or a pin iterator pointing to a pin of a cell instance.
ERROR: cannot find property <name> for device <device_name>.  This error is issued if the named device property is referenced but cannot be found, either because it is missing in the input netlist, or because it is not loaded by PERC Property statements.
ERROR: cannot find the pattern instance name <name> in pattern: <pattern_name>.  This error is issued if the subckt <pattern_name> does not contain the named device.
ERROR: cannot find the pattern net name <name> in pattern: <pattern_name>.  This error is issued if the subckt <pattern_name> does not contain the named net.
ERROR: cannot find the pattern template: <pattern_name>.  This error is issued if <pattern_name> is not a subckt instantiated in the pattern file specified via the PERC Pattern Path statement.
ERROR: cannot increment a single element iterator.  This error is issued if the argument to perc::inc is an iterator pointing to a single element.

**Table 7-73. PERC TVF Errors (cont.)**

Error Message and Description
ERROR: device type is an empty string.  This error is issued if the empty string is specified as a device type.
ERROR: duplicate names in <some> list: <name>.  This error is issued if the same name is specified twice in any list used as an argument to any PERC command.
ERROR: duplicate net type name with different dependencies: <name>.  This error is issued if a net type is defined with different dependencies such that it has to be assigned in multiple phases.
ERROR: invalid -exclude switch, it is allowed only when -break is present.  This error is issued if the command <code>perc::create_net_path</code> is called with the <code>-exclude</code> switch, but without the <code>-break</code> switch.
ERROR: invalid input net: <name>. It is not connected to instance <name> any of the two pins.  This error is issued if the input net to <code>perc::get_instances_in_series</code> or <code>perc::get_other_net_on_instance</code> is not connected to any of the given two pins of the specified instance.
ERROR: invalid <kind> iterator.  This error is issued if the argument to a PERC command is not the correct iterator. This happens if the argument value is not an iterator at all, or it is the different kind of iterator (for instance, a pin iterator when a net iterator is expected), or it is the right kind of iterator, but it has reached the end, that is, its string representation is the empty string.
ERROR: invalid list maximum length <length>. It must be ALL or a positive integer.  This error is issued if the argument to <code>-listMaxLength</code> in calling <code>perc::set_parameters</code> is incorrectly specified.
ERROR: invalid -list and -listPin switches. Cannot specify both.  This error is issued if both <code>-list</code> and <code>-listPin</code> are specified.
ERROR: invalid -listPin switch, it is allowed only when -net is present.  This error is issued if a math command is called with the <code>-listPin</code> switch, but without the <code>-net</code> switch.

**Table 7-73. PERC TVF Errors (cont.)**

Error Message and Description
<p>ERROR: invalid pattern &lt;pattern_name&gt;: duplicate instance names &lt;name&gt;.</p> <p>This error is issued if the subckt &lt;pattern_name&gt; contains more than one device of the same name.</p>
<p>ERROR: invalid pattern &lt;pattern_name&gt;: no path from instance &lt;name&gt; to instance &lt;name2&gt;</p> <p>This error is issued if the subckt &lt;pattern_name&gt; contains a disconnected device.</p>
<p>ERROR: invalid pattern &lt;pattern_name&gt;: no path from net &lt;name&gt; to instance &lt;name2&gt;</p> <p>This error is issued if the subckt &lt;pattern_name&gt; contains a disconnected net.</p>
<p>ERROR: invalid pattern &lt;pattern_name&gt;: no power or ground port found.</p> <p>This error is issued if LVS Power/Ground Name statements are not specified, or none of the port names of the subckt &lt;pattern_name&gt; can be labeled as power or ground.</p>
<p>ERROR: invalid pattern &lt;pattern_name&gt;: non-primitive instance &lt;name&gt; is not allowed.</p> <p>This error is issued if the subckt &lt;pattern_name&gt; contains a cell instance.</p>
<p>ERROR: invalid result maximum count &lt;count&gt;. It must be ALL or a positive integer.</p> <p>This error is issued if the argument to -resultMaxCount in calling perc::set_parameters is incorrectly specified.</p>
<p>ERROR: invalid net type condition list: &lt;list&gt;, &lt;reason&gt;</p> <p>This error is issued if a net type expression has the wrong syntax.</p>
<p>ERROR: invalid net type or type set name: &lt;name&gt;.</p> <p>This error is issued if the named net type or type set is referenced but not defined.</p>
<p>ERROR: invalid net type/set name: &lt;name&gt;. The first char cannot be ‘!’,</p> <p>This error is issued if the named net type or type set starts with a ! character.</p>
<p>ERROR: invalid net type/set name: &lt;name&gt;. The first char cannot be ‘&amp;&amp;’ or ‘  ’.</p> <p>This error is issued if the named net type or type set is named with &amp;&amp; or    characters.</p>
<p>ERROR: invalid -pin argument in calling perc::create_net_path, need at least two pins.</p> <p>This error is issued if there are not enough pins to establish a path across a device.</p>

**Table 7-73. PERC TVF Errors (cont.)**

Error Message and Description
<p>ERROR: invalid pin &lt;name&gt; for device type &lt;type&gt;.</p> <p>This error is issued if devices of the specified type do not have the named pin.</p>
<p>ERROR: invalid -pinAtNet switch, it is allowed only when -net is present.</p> <p>This error is issued if a math command is called with the -pinAtNet switch, but without the -net switch.</p>
<p>ERROR: invalid -pinNetType argument. It must be a list of pairs: {&lt;pin_name_list&gt; &lt;net_type_condition_list&gt; ...}.</p> <p>This error is issued if the -pinNetType argument is not a list consisting of even number of items.</p>
<p>ERROR: invalid -pinPathType argument. It must be a list of pairs: {&lt;pin_name_list&gt; &lt;net_type_condition_list&gt; ...}.</p> <p>This error is issued if the -pinPathType argument is not a list consisting of even number of items.</p>
<p>ERROR: invalid return value &lt;value&gt; for condition proc: &lt;proc&gt;. It must be 0 or 1.</p> <p>This error is issued if the return value of a Tcl proc used as the argument to the -condition switch is neither 0 nor 1.</p>
<p>ERROR: invalid -instanceAlso and -instanceOnly switches. Cannot specify both.</p> <p>This error is issued if both -instanceAlso and -instanceOnly are specified.</p>
<p>ERROR: invalid series pins &lt;pin_1&gt; and &lt;pin_2&gt; for device type &lt;type&gt;.</p> <p>This error is issued if the input pin names to perc::get_instances_in_series or perc::get_other_net_on_instance do not belong to the specified instance.</p>
<p>ERROR: &lt;some&gt; list is empty.</p> <p>This error is issued if any list used as an argument to any PERC command is empty.</p>
<p>ERROR: missing argument in calling &lt;command&gt;: &lt;arg&gt;</p> <p>This error is issued if a required command argument is not provided.</p>
<p>ERROR: more than one -break condition is specified in creating net paths. Only one is allowed.</p> <p>This error is issued if the command perc::create_net_path is called two or more times with the -break switch in a single initialization procedure.</p>

**Table 7-73. PERC TVF Errors (cont.)**

Error Message and Description
<p>ERROR: more than one -cell and/or -cellName argument is specified for net type: &lt;type&gt;. Only one is allowed.</p> <p>This error is issued if the command <code>perc::define_net_type</code> or <code>perc::define_net_type_by_device</code> is called two or more times for the same net type, with the -cell or -cellName switch in a single initialization procedure.</p>
<p>ERROR: more than one rule command is called in the rule check. Only one is allowed.</p> <p>This error is issued if two or more rule check commands are called in a single rule check.</p>
<p>ERROR: property constraint contains invalid &lt;token&gt;.</p> <p>This error is issued if the -property argument is not a string conforming to the constraint syntax.</p>
<p>ERROR: the same name is used to define a net type and a net type set: &lt;name&gt;.</p> <p>This error is issued if a net type shares a name with a type set in a single initialization procedure.</p>
<p>ERROR: unknown argument in calling &lt;command&gt;: &lt;arg&gt;</p> <p>This error is issued if a command argument is not recognized.</p>
<p>ERROR: invalid -adjacentPinNetType or -adjacentPinPathType argument in calling <code>perc::adjacent_count</code>, one and only one must be specified.</p> <p>This error is issued if neither -adjacentPinPathType nor -adjacentPinNetType is specified, or both are specified.</p>
<p>ERROR: wrong -adjacentPinNetType argument. It has to be a pair: {&lt;pin_name_list&gt; &lt;net_type_condition_list&gt;}.</p> <p>This error is issued if the -adjacentPinNetType argument is not a list consisting of two items.</p>
<p>ERROR: wrong -adjacentPinPathType argument. It has to be a pair: {&lt;pin_name_list&gt; &lt;net_type_condition_list&gt;}.</p> <p>This error is issued if the -adjacentPinPathType argument is not a list consisting of two items.</p>
<p>ERROR: wrong arguments in calling &lt;command&gt;.</p> <p>This error is issued if a command is called with incorrect syntax or some argument values are invalid.</p>

**Table 7-73. PERC TVF Errors (cont.)**

Error Message and Description
<p>ERROR: subckt &lt;subckt_name&gt; has already been written to file &lt;file_name&gt;.</p> <p>This error is issued if perc::subckt is called again with &lt;subckt_name&gt; after the same named subckt is built and written to a file using the -write switch.</p>
<p>ERROR: invalid net specified for subckt &lt;subckt_name&gt;: net &lt;net_name&gt; in cell &lt;cell_name&gt; is not connected to any instance in the subckt.</p> <p>This error is issued if perc::subckt is called with the -net switch, but the argument net is not connected to any instance in the subckt.</p>

## Maximum Results and Polygon Segmentation Warnings

**Table 7-74. Maximum Results and Polygon Segmentation Warnings**

Explanation
The following warnings are reported by Calibre nmDRC applications and by the ICrules CHEck DRC command:
<p>WARNING: &lt;number&gt; polygon result(s) segmented in DRC RuleCheck &lt;name&gt;.</p> <p>A polygon result of a rule check required segmentation prior to instantiation in the nmDRC results database. This message is displayed once for each nmDRC rule check with polygon results requiring segmentation. The &lt;name&gt; is the name of the rule check and &lt;number&gt; is the number of polygon results within the rule check which were segmented.</p>
<p>WARNING: Maximum result count of &lt;number&gt; exceeded in DRC RuleCheck &lt;name&gt;.</p> <p>The number of results added to the nmDRC results database for a rule check during the nmDRC run would exceed the maximum number allowed. The &lt;number&gt; is the value of the maximum results parameter and &lt;name&gt; is the name of the rule check.</p>

## Miscellaneous Warnings

**Table 7-75. Miscellaneous Warnings**

Explanation
The following warnings are reported by Calibre nmDRC and by the ICrules CHEck DRc command:
<p>WARNING: Unable to use directory &lt;name&gt; for disk-based layers. Using memory-based layers.</p> <p>Disk file based layers are to be used but nmDRC has a problem creating the specified directory or opening a file in this directory with read/write permission. The run continues with virtual memory based layers used instead.</p>
The following warnings are reported by the ICrules CHEck DRc command:
<p>WARNING: PINS from layer &lt;layer&gt; not read since the hierarchical mode is FLAT.</p> <p>A pins operation is required by the check set and the hierarchical mode of execution is FLAT. The operation generates an empty layer in this event.</p>
<p>WARNING: DRC RuleCheck &lt;name&gt; is prohibited in TOP mode and will not be executed.</p> <p>Reported for each selected nmDRC rule check in the rule check group drc_top_prohibited_checks when the hierarchical mode of the check is TOP.</p>

**Table 7-75. Miscellaneous Warnings (cont.)**

Explanation
WARNING: DRC RuleCheck <name> is prohibited in PEEKED mode and will not be executed. Reported for each selected nmDRC rule check in the rule check group drc_peeked_prohibited_checks when the hierarchical mode of the check is PEEKED.
WARNING: DRC RuleCheck <name> is prohibited in FLAT mode and will not be executed. Reported for each selected nmDRC rule check in the rule check group drc_flat_prohibited_checks when the hierarchical mode of the check is FLAT.
WARNING: DRC interrupt acknowledged; will conclude DRC. Reported when CTRL-S is pressed during execution of CHEck DRc.
WARNING: <name> would not be processed. Reported for every excluded cell name parameter to CHEck DRc which would not have been processed anyway. The most common cause of this warning is a typographic error.

## Output File Open Warnings

**Table 7-76. Output File Open Warnings**

Explanation
The following warnings are reported by Calibre nmDRC applications and by the ICrules CHEck DRC command:
<b>WARNING: Cannot open summary report file &lt;filename&gt; for output.</b> The specified summary report file cannot be opened for write access. The nmDRC run will continue as if a summary report file was not specified.
<b>WARNING: Cannot open database output file &lt;filename&gt; for write access.</b> Reported when the database is being written to binary polygon files but nmDRC is unable to open the output binary polygon file <filename> for write access.
<b>WARNING: Cannot open NET AREA RATIO PRINT file &lt;filename&gt; for output.</b> The file name argument (<filename>) of the net area ratio print operation cannot be opened for write access. No printing occurs in this event.
<b>WARNING: Cannot open DENSITY PRINT file &lt;filename&gt; for output.</b> The file name argument (<filename>) of the print or print only keyword in the density operation cannot be opened for write access. No printing occurs in this event.
<b>WARNING: Cannot open NET AREA RATIO RDB file &lt;filename&gt; for output.</b> The file name argument (<filename>) of the RDB or RDB ONLY keyword in the Net Area Ratio operation cannot be opened for write access. No RDB output occurs in this event.
<b>WARNING: Cannot open DENSITY RDB file &lt;filename&gt; for output.</b> The file name argument (<filename>) of the RDB or RDB ONLY keyword in the Density operation cannot be opened for write access. No RDB output occurs in this event.
<b>WARNING: Cannot open DFM ANALYZE RDB file &lt;filename&gt; for output.</b> The file name argument (<filename>) of the RDB or RDB ONLY keyword in the DFM Analyze operation cannot be opened for write access. No RDB output occurs in this event.
<b>WARNING: Cannot open DFM MEASURE RDB file &lt;filename&gt; for output.</b> The file name argument (<filename>) of the RDB or RDB ONLY keyword in the DFM Measure operation cannot be opened for write access. No RDB output occurs in this event.
<b>WARNING: Cannot open DFM TRANSITION RDB file &lt;filename&gt; for output.</b> The file name argument (<filename>) of the RDB or RDB ONLY keyword in the DFM Transition operation cannot be opened for write access. No RDB output occurs in this event.

**Table 7-76. Output File Open Warnings (cont.)**

Explanation
<p>WARNING: Cannot open LAYOUT INPUT EXCEPTION RDB file &lt;filename&gt; for output.</p> <p>The filename argument (&lt;filename&gt;) of the Layout Input Exception RDB specification statement cannot be opened for write access. No RDB output occurs in this event.</p>
<p>The following is reported by flat Calibre applications:</p> <p>WARNING: Cannot open database input file &lt;filename&gt; for read access.</p> <p>The layout database format is Binary or ASCII and the specified input polygon file could not be opened for read access. The run continues as if the file were empty.</p>
<p>The following is reported by Calibre nmDRC applications: Severity is controlled by <a href="#">Layout Input Exception Severity</a>.</p>
<p>EXCEPTION: Unable to use directory &lt;name&gt; for disk based storage.</p> <p>Disk (file) based geometry storage is to be used but nmDRC has a problem creating the specified directory or opening a file in this directory with read/write permission.</p> <p>Exception name: LAYER_DIRECTORY</p>

## Polygon Flagging Warnings

**Table 7-77. Polygon Flagging Warnings**

Explanation
The following warnings are reported by all Calibre applications and by the ICrules CHEck DRc command:  WARNING: ACUTE angle on layer <layer> at location <point> in cell <name>. Reported for each acute angle flagged due to the specification of the FLAGACUTE option for CHEck DRc or the presence of the Flag Acute YES specification statement for Calibre. The cell name is omitted in Calibre if the layout database format is binary or ASCII. For flat applications, reporting is flat with the point reported in world coordinates. For hierarchical applications, reporting is once per cell with the point reported in cell coordinates.
WARNING: ANGLED edge on layer <layer> from location <point> to location <point> in cell <name>. Reported for each angled edge flagged due to the presence of the Flag Angled YES specification statement for Calibre. The cell name is omitted if the layout database format is BINARY or ASCII. For flat applications, reporting is flat with the point reported in world coordinates. For hierarchical applications, reporting is once per cell with the point reported in cell coordinates.
WARNING: SKEW edge on layer <layer> from location <point> to location <point> in cell <name>. Reported for each skew edge flagged due to the specification of the FLAGSKW option for CHEck DRc or the presence of the Flag Skew YES specification statement for Calibre. The cell name is omitted in Calibre if the layout database format is Binary or ASCII. For flat applications, reporting is flat with the point reported in world coordinates. For hierarchical applications, reporting is once per cell with the point reported in cell coordinates.
WARNING: OFFGRID vertex on layer <layer> at location <point> in cell <name>. Reported for each off-grid vertex flagged due to the specification of the FLAGOFFGRID option for CHEck DRc or the presence of the Flag Offgrid YES specification statement for Calibre. The cell name is omitted in Calibre if the layout database format is Binary or ASCII. For flat applications, reporting is flat with the point reported in world coordinates. For hierarchical applications, reporting is once per cell with the point reported in cell coordinates.
WARNING: Maximum ACUTE angle warning count of 100 has been reached. Reported when the number of warnings generated by acute angle flagging reaches 100—no more warnings are reported for this option.
WARNING: Maximum ANGLED EDGE warning count of 100 has been reached. Reported when the number of warnings generated by angled edge flagging reaches 100 — no more warnings are reported for this option.

**Table 7-77. Polygon Flagging Warnings (cont.)**

Explanation
WARNING: Maximum SKEW EDGE warning count of 100 has been reached. Reported when the number of warnings generated by skew edge flagging reaches 100—no more warnings are reported for this option.
WARNING: Maximum OFFGRID vertex warning count of 100 has been reached. Reported when the number of warnings generated by off-grid vertex flagging reaches 100—no more warnings are reported for this option.
WARNING: Maximum ACUTE angle warning count of <value> has been reached. Reported when the number of warnings generated by acute angle flagging reaches 100 or, in Calibre, the value specified after the optional MAXIMUM WARNINGS keyword in the Flag Acute specification statement — no more warnings are reported for this option.
WARNING: Maximum ANGLED EDGE warning count of <value> has been reached. Reported when the number of warnings generated by angled edge flagging reaches 100 or, in Calibre, the value specified after the optional MAXIMUM WARNINGS keyword in the Flag Angled specification statement — no more warnings are reported for this option.
WARNING: Maximum SKEW EDGE warning count of <value> has been reached. Reported when the number of warnings generated by skew edge flagging reaches 100 or, in Calibre, the value specified after the optional MAXIMUM WARNINGS keyword in the Flag Skew specification statement — no more warnings are reported for this option.
WARNING: Maximum OFFGRID vertex warning count of <value> has been reached. Reported when the number of warnings generated by off-grid vertex flagging reaches 100 or, in Calibre, the value specified after the optional MAXIMUM WARNINGS keyword in the Flag Offgrid specification statement — no more warnings are reported for this option.

## Rule File Errors

**Table 7-78. Rule File Errors**

Explanation
The following fatal errors are reported by all Calibre applications:
ERROR: Cannot determine the LAYOUT SYSTEM from rule file <filename>. The rule file lacks a Layout System specification statement which is required by Calibre.
ERROR: Cannot process the LAYOUT SYSTEM from rule file <filename>. For flat applications, the Layout System specification statement parameter is not GDSII, OASIS, BINARY, or ASCII. For hierarchical applications, the Layout System is not GDSII or OASIS.
ERROR: Cannot determine the LAYOUT PATH from rule file <filename>. The layout database format is GDSII or OASIS but the rule file lacks a Layout Path specification statement, which is required for these layout database formats.
ERROR: Cannot determine the LAYOUT PRIMARY from rule file <filename>. The layout database format is GDSII or OASIS but the rule file lacks a Layout Primary specification statement, which is required for these layout database formats.
ERROR: Incomplete dual database specification in rule file <filename>: <command> without <command>. The rule file contains a Layout System2, Layout Path2, Layout Primary2, or Layout Bump2 specification statement without specification of all four of them.
ERROR: LAYOUT SYSTEM(2) cannot be ASCII or BINARY for dual database capability. Dual database capability is specified but either the Layout System or the Layout System2 is ASCII or Binary. These formats are not supported with dual database capability.
The following fatal error is reported by Calibre nmDRC applications:
ERROR: Cannot determine the DRC RESULTS DATABASE from rule file <filename>. The rule file lacks a DRC Results Database specification statement which is required by Calibre nmDRC.
The following warnings are reported by Calibre nmDRC-H applications:
WARNING: Layout Property Audit is not supported in dual-database mode. The rule file contains a Layout Property Audit specification statement with dual database also specified. The statement is ignored (in other words, OASIS properties are not processed per this statement) in this case.

**Table 7-78. Rule File Errors (cont.)**

Explanation
WARNING: Property input for DFM operations is not supported in dual-database mode. DFM operations in nmDRC-H require properties to be read from the input layout database. This is not supported in dual-database mode.
The following fatal errors are reported by the ICrules CHEck DRc command:
ERROR: There is no rule file loaded. There is no rule file loaded in the Pyxis Layout session.
ERROR: There are no DRC RuleChecks in the rule file. There are no rule check statements in the loaded rule file.
ERROR: There are no DRC RuleChecks selected. There is no check set specified.
ERROR: Cannot establish MASK connectivity of layer <layer> in Continuous DRC RuleCheck <name>. A layer parameter to a nodal dimensional check operation in a selected nmDRC rule check which is an element of the continuous_drc group cannot have its connectivity verified in the mask mode connectivity set.

## Unsupported Functionality Errors and Warnings

**Table 7-79. Unsupported Functionality Errors and Warnings**

Explanation
The following warnings are reported by Calibre nmDRC applications and by the ICrules CHEck DRc command:
WARNING: PATHCHK operation unsupported in DRC applications; output layer will be empty. This operation is not supported in nmDRC applications. Pathchk is used only for ERC checks in LVS connectivity extraction.
WARNING: DEVICE LAYER operation unsupported in DRC applications; output layer will be empty. This operation is not supported in nmDRC applications. The Device Layer operation is used only in Calibre nmLVS/nmLVS-H, Calibre YieldAnalyzer, and ICtrace.
The following warnings are reported by non-nmDRC applications:
WARNING: RDB or RDB ONLY option for NET AREA RATIO only supported in DRC applications. A Net Area Ratio operation with RDB [ONLY] specified is required as part of the flow in a non-nmDRC application. These applications do not support the RDB [ONLY] option. The RDB [ONLY] option is ignored (but the geometric output is generated as usual).

**Table 7-79. Unsupported Functionality Errors and Warnings (cont.)**

Explanation
<p>WARNING: RDB or RDB ONLY option for DENSITY only supported in DRC applications.</p> <p>A Density operation with RDB [ONLY] specified is required as part of the flow in a non-nmDRC application. These applications do not support the RDB [ONLY] option. The RDB [ONLY] option is ignored (but the geometric output is generated as usual).</p>
<p>WARNING: PRINT or PRINT ONLY option for DENSITY only supported in DRC applications.</p> <p>A Density operation with PRINT [ONLY] specified is required as part of the flow in a non-nmDRC application. These applications do not support the PRINT [ONLY] option. The PRINT [ONLY] option is ignored (but the geometric output is generated as usual).</p>
<p>WARNING: NET AREA RATIO PRINT operation only supported in DRC applications.</p> <p>A Net Area Ratio Print operation is required as part of the flow in a non-nmDRC application. These applications do not support the Net Area Ratio Print operation. An empty layer is output.</p>
<p>WARNING: DFM ANALYZE operation only supported in Calibre DRC applications; output layer will be empty.</p> <p>A DFM Analyze operation is required as part of the flow in a non-Calibre nmDRC application. The output layer will be empty.</p>
<p>WARNING: DFM MEASURE operation only supported in Calibre DRC applications; output layer will be empty.</p> <p>A DFM Measure operation is required as part of the flow in a non-Calibre nmDRC application. The output layer will be empty.</p>
<p>WARNING: DFM TRANSITION operation only supported in Calibre DRC applications; output layer will be empty.</p> <p>A DFM Transition operation is required as part of the flow in a non-Calibre nmDRC application. The output layer will be empty.</p>
The following warnings are reported by ICverify applications:
<p>WARNING: INSIDE CELL operation unsupported in ICverify applications; output layer(s) will be empty.</p> <p>An inside cell operation is required in an ICverify application. This operation is only supported in Calibre applications. All output layers are empty.</p>
<p>WARNING: NOT INSIDE CELL operation unsupported in ICverify applications; output layer(s) will be copies.</p> <p>Issued when an Not Inside Cell operation is required in an ICverify application. This operation is only supported in Calibre applications. All output layers will be copies of the input layer.</p>

**Table 7-79. Unsupported Functionality Errors and Warnings (cont.)**

Explanation
<p>WARNING: EXTENT CELL operation unsupported in ICverify applications; output layer will be empty.</p> <p>An Extent Cell operation is required in a flat application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: EXTENT DRAWN operation unsupported in ICverify applications; output layer will be empty.</p> <p>An Extent Drawn operation is required in a flat application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: OPCLINEEND operation unsupported in ICverify applications; output layer will be empty.</p> <p>An Opclineend operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: OPCSBAR operation unsupported in ICverify applications; output layer will be empty.</p> <p>An Opcobar operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: OPCBIAS operation unsupported in ICverify applications; output layer will be empty.</p> <p>An Opcbias operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: LITHO operation unsupported in ICverify applications; output layer will be empty.</p> <p>A Litho operation is required in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: FRACTURE operation unsupported in ICverify applications; output layer will be empty.</p> <p>A fracture operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: MDP operation unsupported in ICverify applications; output layer will be empty.</p> <p>An MDP operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>

**Table 7-79. Unsupported Functionality Errors and Warnings (cont.)**

Explanation
<p>WARNING: MDPMERGE operation unsupported in ICverify applications; output layer will be empty.</p> <p>An MDPMerge operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: MDPSTAT operation unsupported in ICverify applications; output layer will be empty.</p> <p>An MDPstat operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: MDPVERIFY operation unsupported in ICverify applications; output layer will be empty.</p> <p>An MDPverify operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: TDDRC operation unsupported in ICverify applications; output layer will be empty.</p> <p>A TDDRC operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: DFM ANALYZE operation unsupported in ICverify applications; output layer will be empty.</p> <p>A DFM Analyze operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: DFM MEASURE operation unsupported in ICverify applications; output layer will be empty.</p> <p>A DFM Measure operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>WARNING: DFM TRANSITION operation unsupported in ICverify applications; output layer will be empty.</p> <p>A DFM Transition operation is required as part of the flow in an ICverify application. This operation is only supported in Calibre applications. The output layer will be empty.</p>
<p>The following warnings are reported by Calibre applications:</p> <p>WARNING: PINS operation unsupported in Calibre applications; output layer will be empty.</p> <p>This operation is only applicable to ICverify.</p>

**Table 7-79. Unsupported Functionality Errors and Warnings (cont.)**

Explanation
<p>WARNING: PORTS operation unsupported in Calibre applications; output layer will be empty. This operation is only applicable to ICVerify.</p>
<p>WARNING: TOPEX operation unsupported in Calibre applications; output layer will be empty. This operation is only applicable to ICVerify.</p>
<p>The following warnings are reported by flat nmDRC:</p> <p>WARNING: EXTENT CELL ... WITH MATCH unsupported in flat CALIBRE applications. WITH MATCH option ignored. The WITH MATCH option of the Extent Cell operation is only supported in hierarchical Calibre applications and is ignored in flat applications.</p>
<p>WARNING: INSIDE CELL ... WITH MATCH unsupported in flat CALIBRE applications. WITH MATCH option ignored. The WITH MATCH option of the (Not) Inside Cell operation is only supported in hierarchical Calibre applications and is ignored in flat applications.</p>
<p>WARNING: DRC CHECK MAP AREF specification for DRC RuleCheck &lt;name&gt; not supported in CALIBRE DRC-F. The AREF keyword is specified in a DRC Check Map specification statement for a nmDRC rule check in Calibre nmDRC-F. AREF output for arrayed rectangles is only supported in Calibre nmDRC-H. For Calibre nmDRC-F, rectangles are output without AREF structure. The name of the affected nmDRC rule check appears in the message.</p>
<p>WARNING: DRC CHECK MAP TEXTTAG specification for DRC RuleCheck &lt;name&gt; not supported in CALIBRE DRC-F. The TEXTTAG keyword is specified in a DRC Check Map specification statement for a DRC rule check in flat nmDRC. TEXTTAG output is only supported in nmDRC-H and is otherwise ignored.</p>
<p>WARNING: DRC CHECK MAP AUTOREF specification for DRC RuleCheck &lt;name&gt; not supported in CALIBRE DRC-F. The AUTOREF keyword is specified in a DRC Check Map specification statement for a DRC rule check in flat nmDRC. AUTOREF output is only supported in nmDRC-H and is otherwise ignored.</p>
<p>WARNING: DRC CHECK MAP PROPERTIES specification for DRC RuleCheck &lt;name&gt; not supported in CALIBRE DRC-F. The PROPERTIES keyword is specified in a DRC Check Map specification statement for a DRC check in Calibre nmDRC. PROPERTIES output is only supported in nmDRC-H. For flat operation, PROPERTIES is ignored. The name of the affected DRC check is placed into the message.</p>

**Table 7-79. Unsupported Functionality Errors and Warnings (cont.)**

Explanation
The following errors are reported by nmDRC.
ERROR: CBLOCK OASIS DRC results database not supported with PIPE. The CBLOCK keyword is not supported with PIPE in DRC Results Database.
ERROR: STRICT OASIS DRC results database not supported with PIPE. The STRICT keyword is not supported with PIPE in DRC Results Database.
ERROR: ASCII DRC results database not supported with PIPE. ASCII format is not supported with PIPE in DRC Results Database.
ERROR: Binary DRC results database not supported with PIPE. The BINARY keyword is not supported with PIPE in DRC Results Database.

# Chapter 8

## Rule File Examples

---

This chapter is a compendium of sample rule files, nmDRC rule checks, and layer derivation operations. These examples show how design rule checks can be used in commonly-encountered situations in layout verification.

Sample Rule File . . . . .	2069
Width Checks . . . . .	2073
Spacing Checks . . . . .	2076
Enclosure and Extension Checks . . . . .	2089
Contact Checks . . . . .	2096
Deriving Layers . . . . .	2105
Other nmDRC Checks . . . . .	2109
Device Property Calculation Examples . . . . .	2113
ICVerify Applications . . . . .	2115

### Sample Rule File

```
//////////  
// DRAWN LAYERS  
//////////  
  
layer substrate      0  
layer pwell         1  
layer oxide          2  
layer res            3  
layer poly           4  
layer nplus          5  
layer pplus          6  
layer contact        7  
layer metall1        8  
layer via             9  
layer metal2        10  
layer ccl            35  
layer dfr            38  
  
//////////  
// DEFINE BOOLEAN LAYERS  
//////////  
  
nsub = substrate NOT pwell  
// diffused resistor layers  
dfcnt  = contact INSIDE dfr  
dfcnto = SIZE dfcnt BY 3  
dfrs   = dfr NOT dfcnto  
dpres  = dfrs AND pplus  
dnres  = dfrs AND nplus
```

## Rule File Examples

### Sample Rule File

---

```
// poly resistor AND interconnect poly layers
resist = SIZE res BY -1
trespl = poly CUT resist
unused = trespl OUTSIDE contact
tmpoly = trespl NOT unused
gresist = resist AND tmpoly
rpoly = gresist AND poly
ipoly = poly NOT rpoly

// p-diffusion layers
oxsub = oxide AND nsub
narea = oxsub NOT ccl
pgate = narea AND poly
pox = oxide AND pplus
psdt = pox NOT poly
psd = psdt NOT dfrs // p-diffusion
// n-diffusion layers
oxwell = oxide AND pwell
parea = oxwell NOT ccl
ngate = parea AND poly
nox = oxide AND nplus
nsdt = nox NOT poly
nsd = nsdt NOT dfrs // n-diffusion
shtcnt = AREA contact > 6.1 < 6.2

///////////////////////////////
// SIMPLE nmDRC CHECKS
///////////////////////////////

GROUP mask_check // all the DRC checks for mask-level data
poly_width poly_spacing dr2w dr2s dr3 dr5
dr6w dr7 dr11pp dr11np dr12 dr13 dr14 dr17np
dr17pp minimum_contact dr20 dr21 dr23 dr26
dr28 dr29 dr30

poly_width {
@Poly width must be 0.25
INTERNAL poly < 0.25 ABUT < 90 SINGULAR
}

poly_spacing {
@Poly spacing must be 0.25
EXTERNAL poly < 0.25 ABUT < 90 SINGULAR
}

dr2w { INTERNAL oxide < 0.35 ABUT < 90 SINGULAR}

dr2s { EXTERNAL oxide < 0.35 ABUT < 90 SINGULAR}

dr3 { EXTERNAL pwell < 1.0 ABUT < 90 SINGULAR SPACE}

dr5 { EXTERNAL poly oxide < 0.25 ABUT < 90 SINGULAR}

dr6w { INTERNAL metall1 < 0.25 ABUT < 90 SINGULAR}

dr7 { EXTERNAL metal2 < 0.25 ABUT < 90 SINGULAR}

dr11pp { EXTERNAL pplus contact < 0.25 ABUT < 90 SINGULAR}
```

```
dr11np { EXTERNAL nplus contact < 0.25 ABUT < 90 SINGULAR}
```

```
dr12 {  
    shtcnt = AREA contact > 6.1 < 6.2  
    EXTERNAL shtcnt poly < 1.675 ABUT < 90 SINGULAR  
}
```

```
dr13 { EXTERNAL oxide poly < 0.25 ABUT < 90 SINGULAR}
```

```
dr14 { ENCLOSURE oxide poly < 0.25 ABUT < 90 SINGULAR}
```

```
dr17np { ENCLOSURE oxide nplus < 0.25 ABUT < 90 SINGULAR}
```

```
dr17pp { ENCLOSURE oxide pplus < 0.25 ABUT < 90 SINGULAR}
```

```
minimum_contact {  
    x = NOT RECTANGLE contact == 0.25 BY == 0.25  
    @contacts must be either exactly 0.25 by 0.25 or  
    @they must be exactly 0.25 by 1.0  
    NOT RECTANGLE x == 0.25 BY == 1.0  
}
```

```
dr20 { EXTERNAL contact oxide < 0.25 ABUT < 90 SINGULAR}
```

```
dr21 {  
    pgate = poly AND oxide  
    EXTERNAL contact pgate < 0.25 ABUT < 90 SINGULAR  
}
```

```
dr23 { ENCLOSURE contact metall1 < 0.1 ABUT == 0 OVERLAP}
```

```
dr26 { EXTERNAL via oxide < 0.25 ABUT < 90 SINGULAR}
```

```
dr28 {  
    ENCLOSURE via metall1 < 0.1 ABUT == 0 OVERLAP  
    ENCLOSURE via metal2 < 0.1 ABUT == 0 OVERLAP  
}
```

```
dr29 { NOT RECTANGLE via == 0.05 BY == 0.05}
```

```
dr30 { INTERNAL metal2 < 0.25 ABUT < 90 SINGULAR}
```

```
//////////////////////////////  
// CONNECT OPERATIONS  
//////////////////////////////
```

```
CONNECT psd pwell  
// note that pseudo-contacts are not needed for tie-downs  
CONNECT nsd nsub  
CONNECT metall1 ipoly nsd psd by contact  
// connects metall1 to the first of ipoly nsd psd  
CONNECT metal2 metall1 by via MASK
```

```
//////////////////////////////  
// Intentional Device Definitions for Mask LVS
```

## Rule File Examples

### Sample Rule File

---

```
//////////  
DEVICE mn ngate ipoly nsd nsd pwell [0] // nmos transistors  
DEVICE mp pgate ipoly psd psd nsub [0] // pmos transistors  
DEVICE r(pl) rpoly ipoly ipoly [ 20000 ] // poly resistors  
DEVICE r(dp) dpres psd psd [170] // diffused p-type resistors  
DEVICE r(dn) dnres nsd nsd [65] // diffused n-type resistors
```

# Width Checks

This section contains examples of nmDRC width checks.

## Minimum width

```
METAL_WIDTH {
    //Metal width must be greater than or equal to 3 microns
    INTERNAL metal < 3 ABUT < 90 SINGULAR
    // Output into results database
}
```

## Minimum gate width

```
Gate_Width { @ Minimum: ^gate_width um
N = NGate COINCIDENT EDGE Poly
P = PGate COINCIDENT EDGE Poly
LENGTH N < gate_width
LENGTH P < gate_width
}
```

## Width on long objects

```
METAL_WIDTH {
    //Metal width must be greater than or equal to 3 microns
    //except where metal length exceeds 5 microns; in that
    //case metal width must be greater than or equal to 4 microns.
    INTERNAL metal < 3 // Output into results database
    long_metal = metal LENGTH > 5 // Layer definition
    INTERNAL metal long_metal < 4 // Output into results database
}
```

## Thin metal polygons

```
// Find all metal polygons having a width somewhere less than 3 microns.
x = internal [metal] < 3
thin_metal_polygons { metal with edge x}
//shows whole polygons that have thin metal somewhere in their
//length, not just at the point of error
```

## Gate width

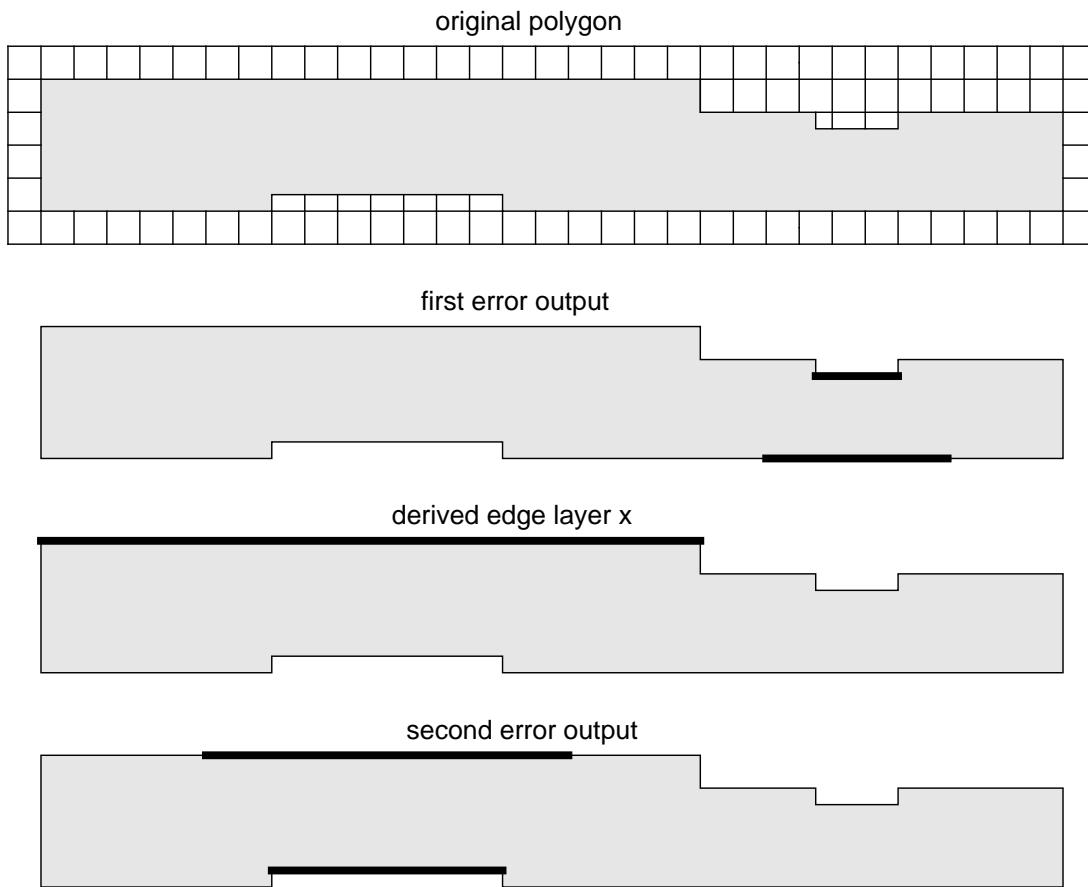
```
narrow_gate {
    @ Gate width, including bent gates, must be at least
    @ 4 microns. Flag gates themselves, not gate edges.
    gate_edge = gate coincident edge poly
    short_edge = length gate_edge < 4
    gate with edge short_edge
}
```

## Width check with length dependencies

```
// SIMPLE METAL RULES CHECK
```

```
// Notice in the following example that the layer of origin of
// the two input layers in the second internal check is the same.
METAL_WIDTH_CHECK {
    @ The width of metal must be greater than or equal to 3
    @ microns except where metal edge length is greater than or
    @ equal to 20 microns, in which case the metal width must
    @ be greater than or equal to 4 microns.
    internal metal < 3//First Error Output
    x = length metal >= 20// Derived Edge layer X
    internal metal x < 4 // <-- NOTE same layer of origin.
}
```

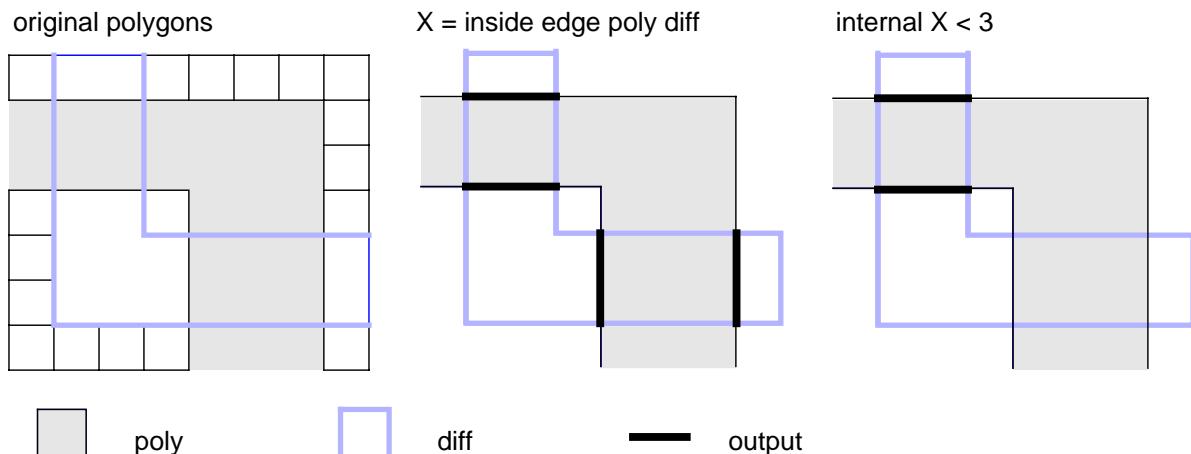
**Figure 8-1. METAL\_WIDTH\_CHECK**



## Minimum gate length check

```
gate_dimension_rule {
    @ Transistor gate dimension (length) between poly edges
    @ must be at least 3 microns.
    X = poly inside edge diff//creates derived layer X
    internal X < 3//error output
}
```

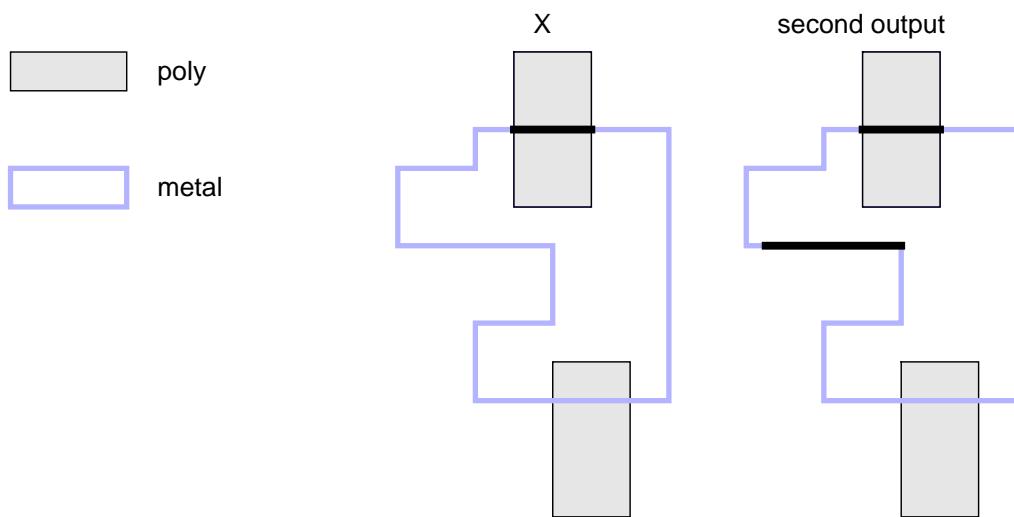
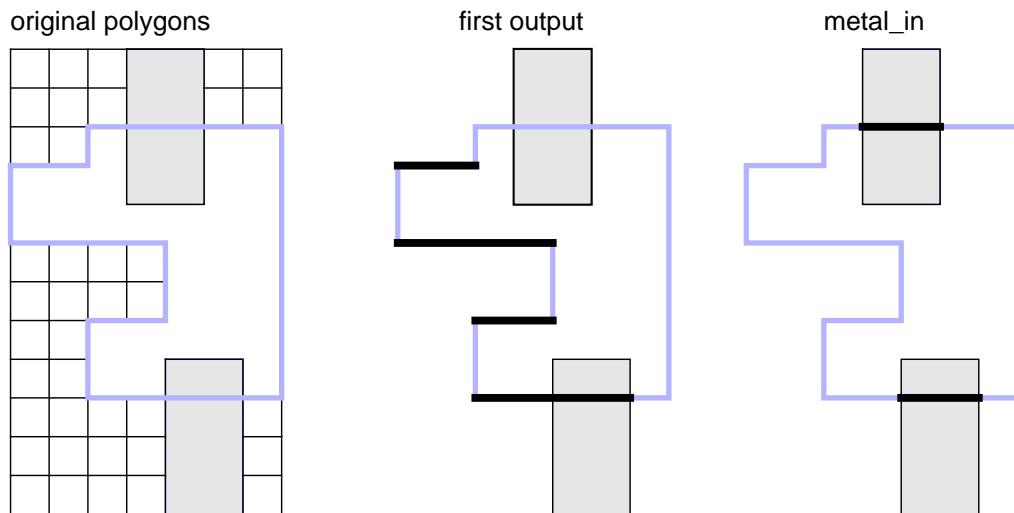
**Figure 8-2. gate\_dimension\_rule**



### Width with second layer overlap dependencies

```
// METAL WIDTH IN POLY OVERLAP CHECK
METAL004 {
    @ The width of metal must be greater than or equal to 3
    @ microns except where poly overlaps metal by more than 1
    @ micron, in which case the metal width must be greater than
    @ or equal to 4 microns.
    internal metal < 3
    metal_in = metal inside edge poly
    X = internal (metal_in) poly <= 1
    internal metal X < 4 }
```

**Figure 8-3. Width With Overlap**



## Wide metal1 regions

```
// Find metall1 regions where the width is greater than 4 um:  
wide_metal1{  
    fat_metal1a = SIZE metall1 BY -2 // Width <= 4 goes away.  
    fat_metal1 = SIZE fat_metal1a BY 2  
    // Restore metal that did not disappear.  
    copy fat_metal1  
}
```

## Spacing Checks

This section contains examples of nmDRC spacing checks.

## Minimum metal spacing (same layer)

```
Metal_Spacing { @ Minimum spacing: ^metal_spacing um
EXTERNAL metal < metal_spacing ABUT < 90 SINGULAR SQUARE
}
```

## Spacing between varying-width objects

```
Shrunk_Metall1 = SIZE Metall1 BY metall1_shrink
Wide_Metall1   = SIZE Shrunk_Metall1 BY metall1_enlarge

Wide_M1_Edges  = Metall1 COINCIDENT EDGE Wide_Metall1
Thin_M1_Edges  = Metall1 NOT COINCIDENT EDGE Wide_Metall1

Wide_M1_Spacing {
@ (width >= ^wide_metal1_width um) spacing:
@ ^wide_metal1_spacing um
EXTERNAL Wide_M1_Edges < wide_metal1_spacing
}

Wide_to_Thin_M1_Spacing {
@ over distances of ^wide_metal1_length um:
@ ^wide_metal1_spacing um
EXTERNAL Wide_M1_Edges Thin_M1_Edges < wide_metal1_spacing
PARALLEL PROJECTING > wide_metal1_length
}
```

## Minimum spacing gate to well tap

```
// Size Tap CT1 by chunks in these two checks, rather than
// in one jump of the full rule. This prevents the check
// from jumping across a well and picking up a well tie
// there. tap_size_increment = 1/5 tap_gate_spacing
NTap_PGate_Spacing { @ Maximum: ^tap_gate_spacing
X1 = SIZE Tap_CT1 BY tap_size_increment
X2 = X1 AND NWell
X3 = SIZE X2 BY tap_size_increment
X4 = X3 AND NWell
X5 = SIZE X4 BY tap_size_increment
X6 = X5 AND NWell
X7 = SIZE X6 BY tap_size_increment
X8 = X7 AND NWell
X9 = SIZE X8 BY tap_size_increment
gate NOT X9
}
```

## Well tap to source-drain spacing

```
Tap_to_Act_Spacing {
@ Minimum spacing same potential: 0 um.
@ Different potentials: ^tap_spacing um
Good = EXTERNAL Tap Active == 0 ABUT == 0 CONNECTED REGION
Bad = EXTERNAL Tap Active < tap_spacing ABUT == 0 REGION
Bad NOT Good
}
```

## Spacing between any edge on same layer

```
METAL_SPACING { EXTERNAL metal < 5 ABUT < 90 SINGULAR }
```

## Spacing between edges on same polygon

```
METAL_NOTCH { metal external < 3.5 notch abut < 90 singular
// Check spacing in the same polygon.
}
```

## Spacing between different polygons on same layer

```
METAL_SPACING { metal external < 4 space abut < 90 singular
// Spacing between polygons, same layer.
}
```

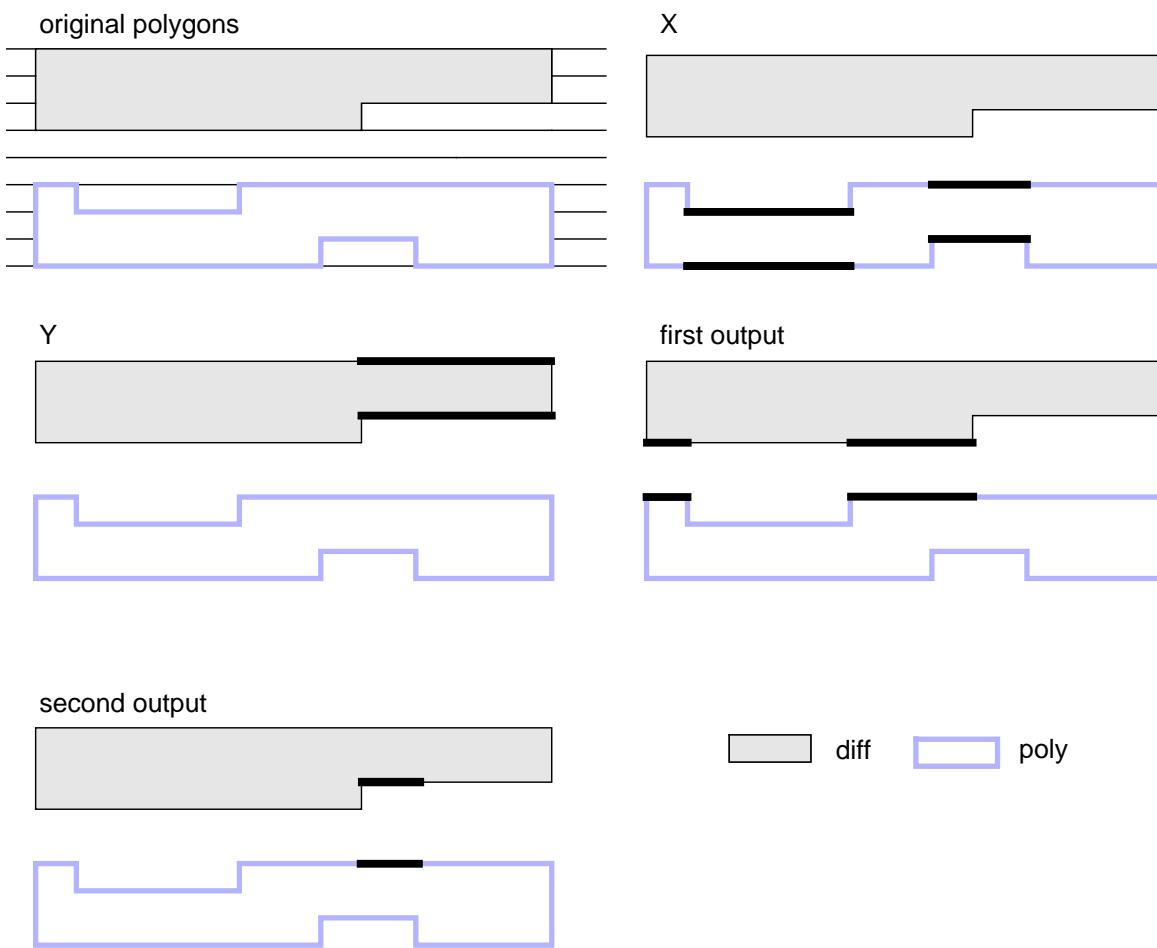
## Spacing metal to wide metal on single layer

```
METAL_TO_FAT_METAL_SPACING {
@ Metal to fat metal spacing must be 4 microns.
@ DO NOT check notches!
@ (Fat metal is any metal fatter than 5 microns).
x = SIZE metal BY -2.5
y = SIZE x BY 2.5
// Chunks of fat metal (typical technique).
// Layer z will get edges from the original metal layer
// which are on fat pieces. This gets rid of edges on y
// which were not originally on metal (eliminating false
// errors). Also, since z and metal have the same layer of
// origin, we can stipulate spacing only in the
// external operation below since it is really a one-layer
// dimensional check operation.
z = metal COINCIDENT EDGE y// Metal edges on fat metal.
EXT z metal < 4 SPACE// Check spacing only (no notch).
}
```

## Poly-diffusion separation

```
// POLY AND DIFFUSION SEPARATION CHECK
POLY_DIFF_SEP {
@ Separation of poly and diffusion must be greater than
@ or equal to 3 microns except where both poly width and
@ diffusion width is less than 3 microns; in that case, the
@ separation must be greater than or equal to 4 microns.
    X = internal [poly] < 3
    Y = internal [diff] < 3
    external poly diff < 3
    external X Y < 4
}
```

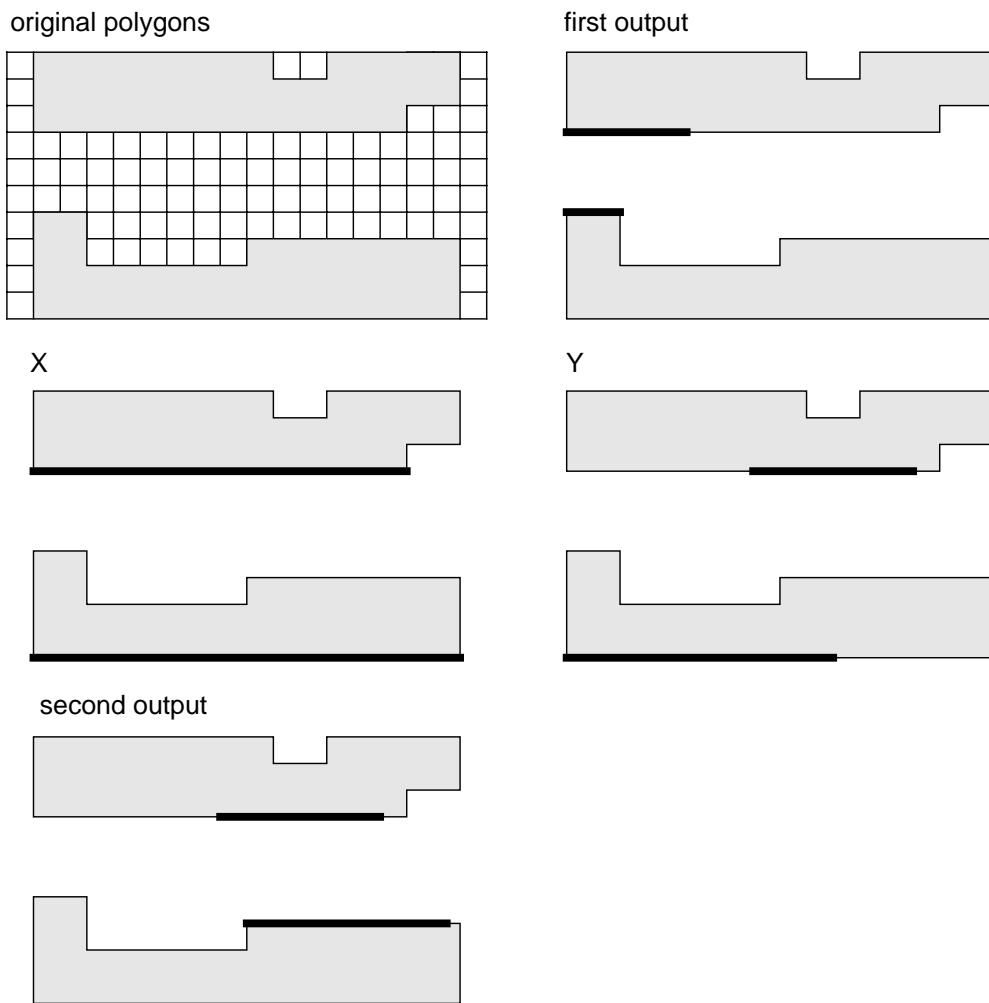
**Figure 8-4. POLY\_DIFF\_SEP**



### Spacing with width-to-length dependencies

```
METAL_SPACING {
    @ Metal to metal spacing must be greater than or equal to 4
    @ microns except where (1) the metal edge length is greater
    @ than 10 microns and, (2) the width in case (1) is less than
    @ 3 microns. In the latter case, the metal spacing to edges
    @ satisfying both (1) and (2) must be greater than or equal to
    @ 5 microns.
    external metal < 4
    X = metal length > 10
    Y = internal metal [X] < 3
    external metal Y < 5}
```

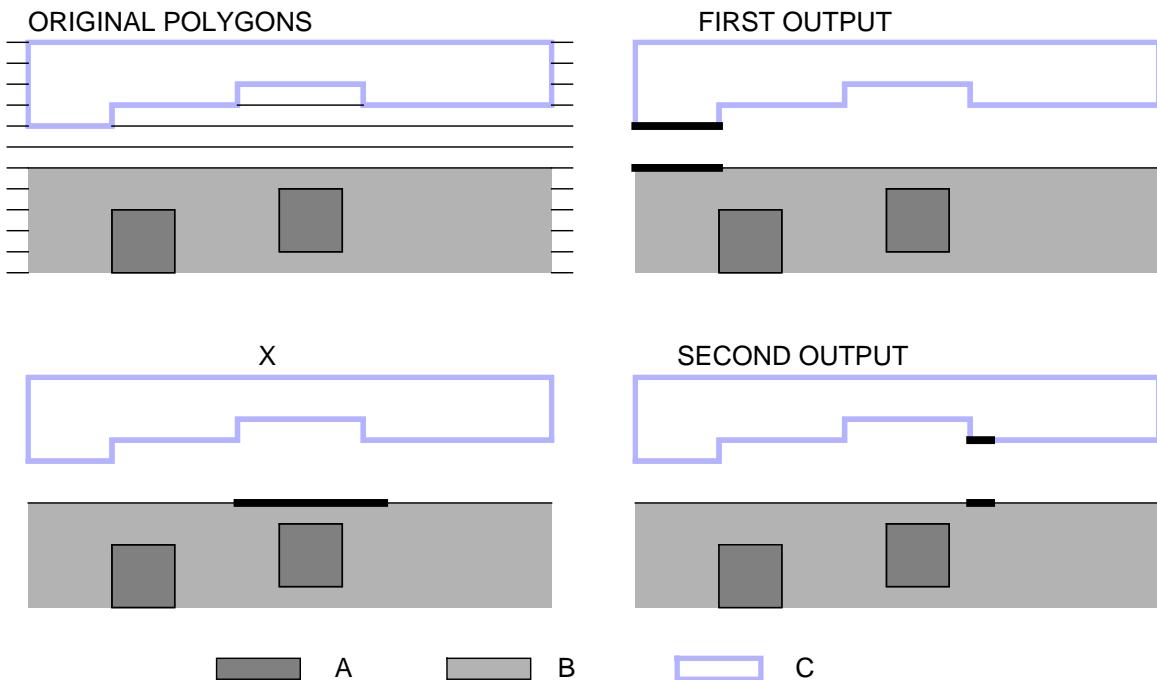
**Figure 8-5. METAL\_SPACING**



### Corner effect separation

```
BC_CORNER_ERROR {
  @ Separation of B and C must normally be greater than or equal
  @ to 3 microns. When A is inside B and closer than 2 microns,
  @ then the spacing between B and C at that point must be
  @ increased to 4 microns and this spacing must extend for two
  @ microns beyond each corner of A.
  external B C < 3 opposite
  X = enclosure A [B] < 2 square
  externalX C < 4 opposite}
```

**Figure 8-6. BC\_CORNER\_ERROR**



## Spacing with simple halation

```
Metal_Halation {
    // METAL SPACING WITH SIMPLE HALATION EXCEPTION
    @ Metal spacing must be 3 microns except between parallel
    @ edges whose common projective edge length is greater than or
    @ equal to 20 microns; in that case, the spacing must be 4
    @ microns.
    EXT metal < 3 ABUT < 90 SINGULAR// Normal rule
    // Halation exception follows. Note the use of the OPPOSITE
    // metric for more precise measurement and the interval
    // constraint to avoid duplicating errors from the normal
    // rule.
    EXT metal >= 3 < 4 PARALLEL ONLY OPPOSITE PROJECTING >= 20}
```

## Spacing between angled edges

```
Poly_Spacing_both_angled {
    EXT poly < 3 ANGLED == 2
    // Measure two edges only if both are angled.
}

Poly_Spacing_oneplus_angled {
    EXT poly < 3 ANGLED
    // Measure two edges only if at least one is angled.
}
```

```
Poly_Spacing_oneonly_angled {
    EXT poly < 3 ANGLED == 1
    // Measure two edges only if exactly one is angled.
}

Poly_Spacing_neither_angled {
    EXT poly < 3 ANGLED == 0
    // Measure two edges only if neither one is angled.
}
```

## Spacing one layer corner-to-corner

```
poly_spacing {
    @ poly spacing = 2 with the square metric applied for
    @ corner to edge and corner to corner spacings.
    EXT poly < 2 NOT CORNER
    EXT poly < 2 CORNER SQUARE
}

poly_spacing {
    @ poly spacing = 2 with the square metric applied
    @ to corner-to-corner spacings.
    @ Corner-to-edge spacing is 2.5.
    EXT poly < 2 NOT CORNER
    EXT poly < 2 CORNER TO CORNER     SQUARE
    EXT poly < 2.5 CORNER TO EDGE
}
```

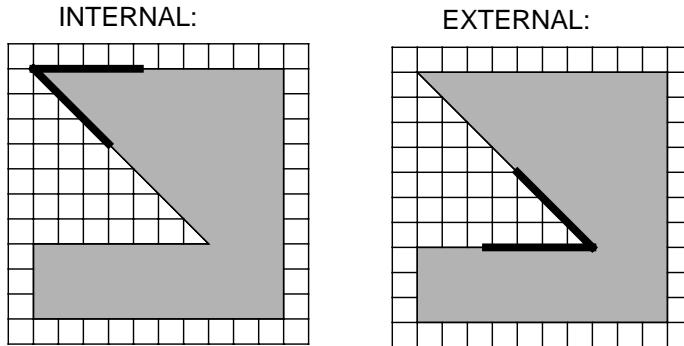
## Spacing two layer corner-to-corner

```
poly_spacing {
    @ poly to diff spacing = 2 with the square metric applied
    @ for corner to edge and corner to corner spacings.
    EXT poly diff < 2 NOT CORNER
    EXT poly diff < 2 CORNER SQUARE
}
```

## Spacing of specified range of angles on a single layer

```
"ACUTE ANGLE LENGTH CHECK" {
    @ Measure the length of sides of an single layer acute angle
    @ for lengths > 3
    INTERNAL layer1 < 3 ABUT < 90
    EXTERNAL layer1 < 3 ABUT < 90}
```

**Figure 8-7. ACUTE ANGLE LENGTH CHECK**



### Spacing of specified range of angles on two layers

```

ACUTE_TWO_LAY1 {
    @ Catch acute angles between polygons on different layers in a
    @ two-layer external check.
    external metal poly < 3 abut > 0 < 90
    //0 was excluded from the abut constraint so that edges which
    //are coincident outside would not also be output.
}

ACUTE_TWO_LAY2 {
    @ Additional output (in addition to acute angles) of
    @ coincident outside edges
    external metal poly < 3 abut < 90
}

ACUTE_TWO_LAY3 {
    @ ONLY coincident outside edges to be additionally output
    external metal poly < 3 abut == 0
}

```

### Spacing with third layer dependencies

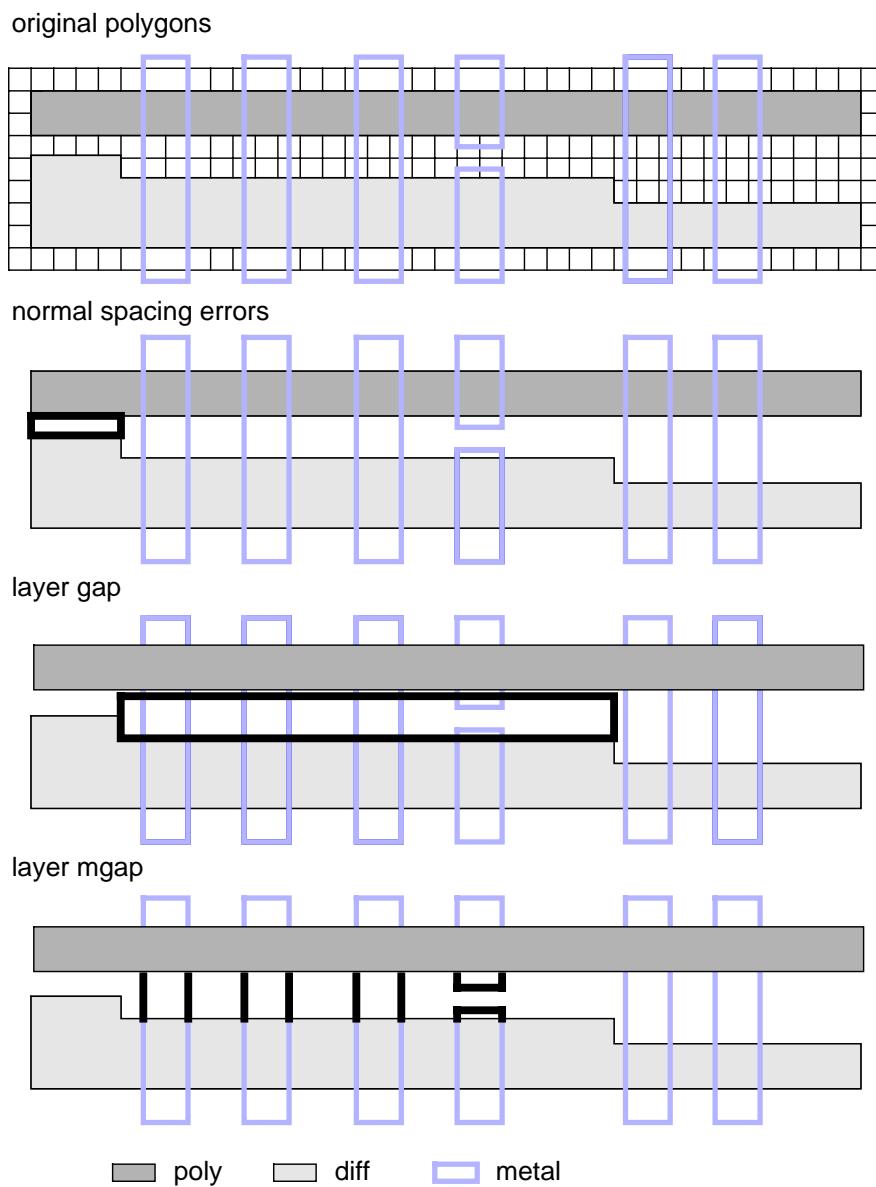
```

POLY_DIFF_SPACING {
    @ This rule uses both polygon-directed output and the
    @ intersecting only filter.Poly to diff spacing is normally 1
    @ micron. However, if two metal wires completely cross the
    @ area between poly and diff, are perpendicular to both poly
    @ and diff, and are closer than 3 microns apart, then the
    @ spacing between poly and diff BETWEEN the metal wires must
    @ be increased to 2 microns.
    // Normal spacing rule.
    EXTERNAL poly diff < 1 REGION OPPOSITE
    // Form the regions between poly and diff where the spacing is
    // less than 2 microns. Call this region the "gap".
    gap = EXTERNAL poly diff >= 1 < 2 REGION OPPOSITE
    // Select all metal edges which span the "gap" and are
    // perpendicular to both poly and diff. Note that the metal
    // wires must completely span the gap.
    mgap = metal INSIDE EDGE gap// Metal edges in the gap
}

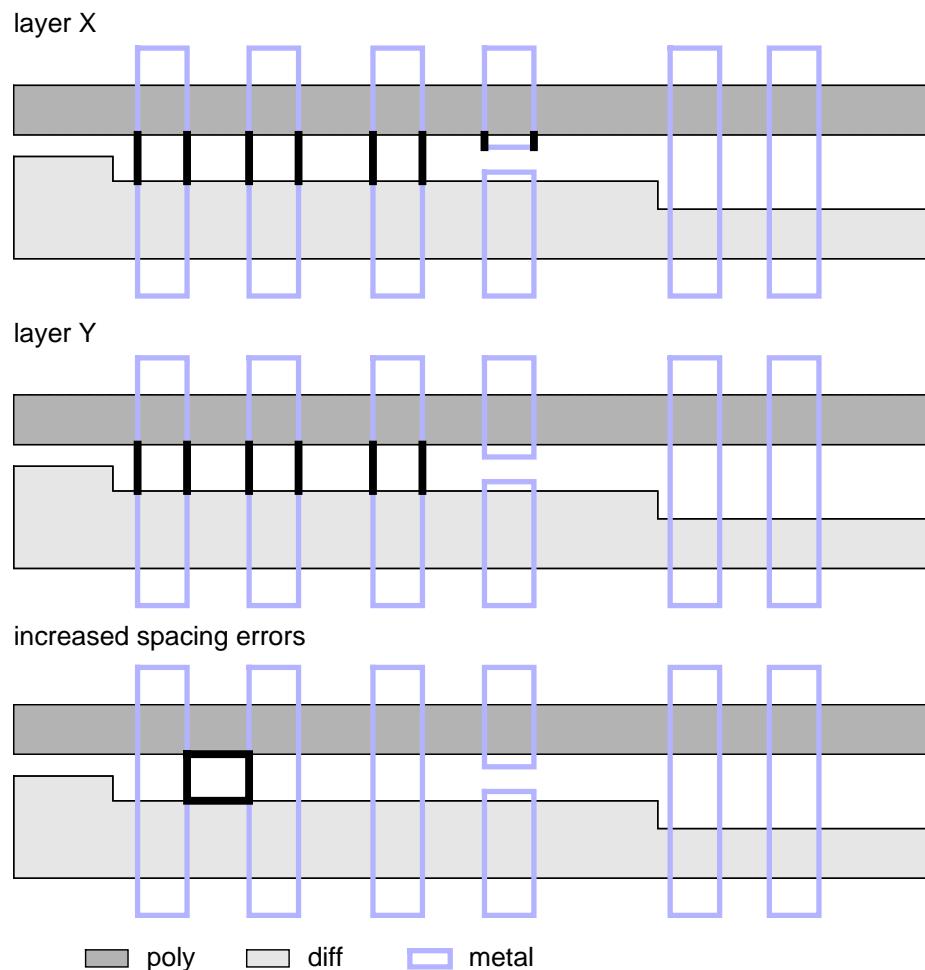
```

```
X = EXTERNAL [mgap] poly < 2 ABUT == 90 INTERSECTING ONLY
Y = EXTERNAL [X] diff < 2 ABUT == 90 INTERSECTING ONLY
// The errors can now be flagged between the edges in layer Y
// since these edges completely span the gap perpendicular to
// it.
EXTERNAL Y < 3 REGION OPPOSITE
}
```

**Figure 8-8. POLY\_DIFF\_SPACING (part 1)**



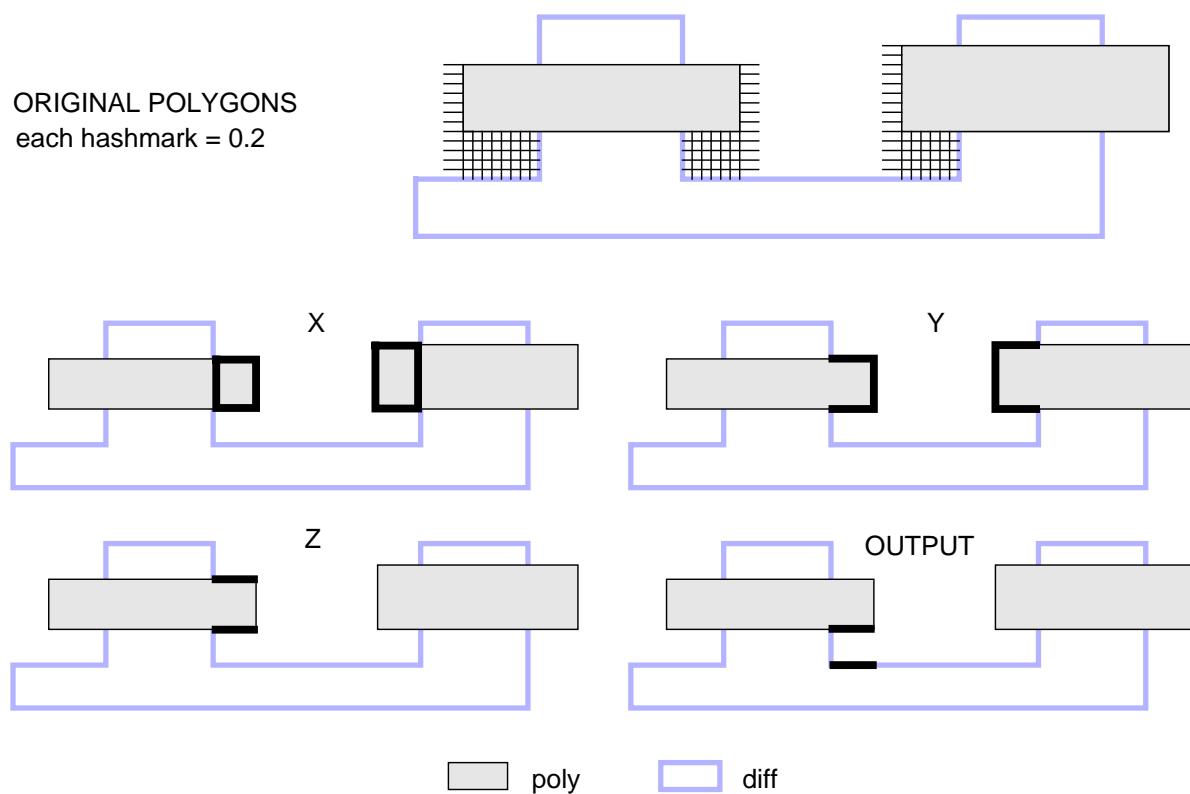
**Figure 8-9. POLY\_DIFF\_SPACING (part 2)**



### Poly-diff corner spacing

```
Rule7.4 {
  @ Poly spacing to source/drain corner must be 1.2 whenever the
  @ poly gate length is less than 1.6 and poly extension over
  @ gate is less than 1.4.
  x = ENC diff poly < 1.4 REGION OPPOSITE// Small extensions.
  y = x NOT COINCIDENT EDGE diff// Sans gate edges
  z = INT [y] < 1.6 OPPOSITE// Small extensions at short gates.
  EXT z diff < 1.2 OPPOSITE// Errors
}
```

**Figure 8-10. Rule7.4**



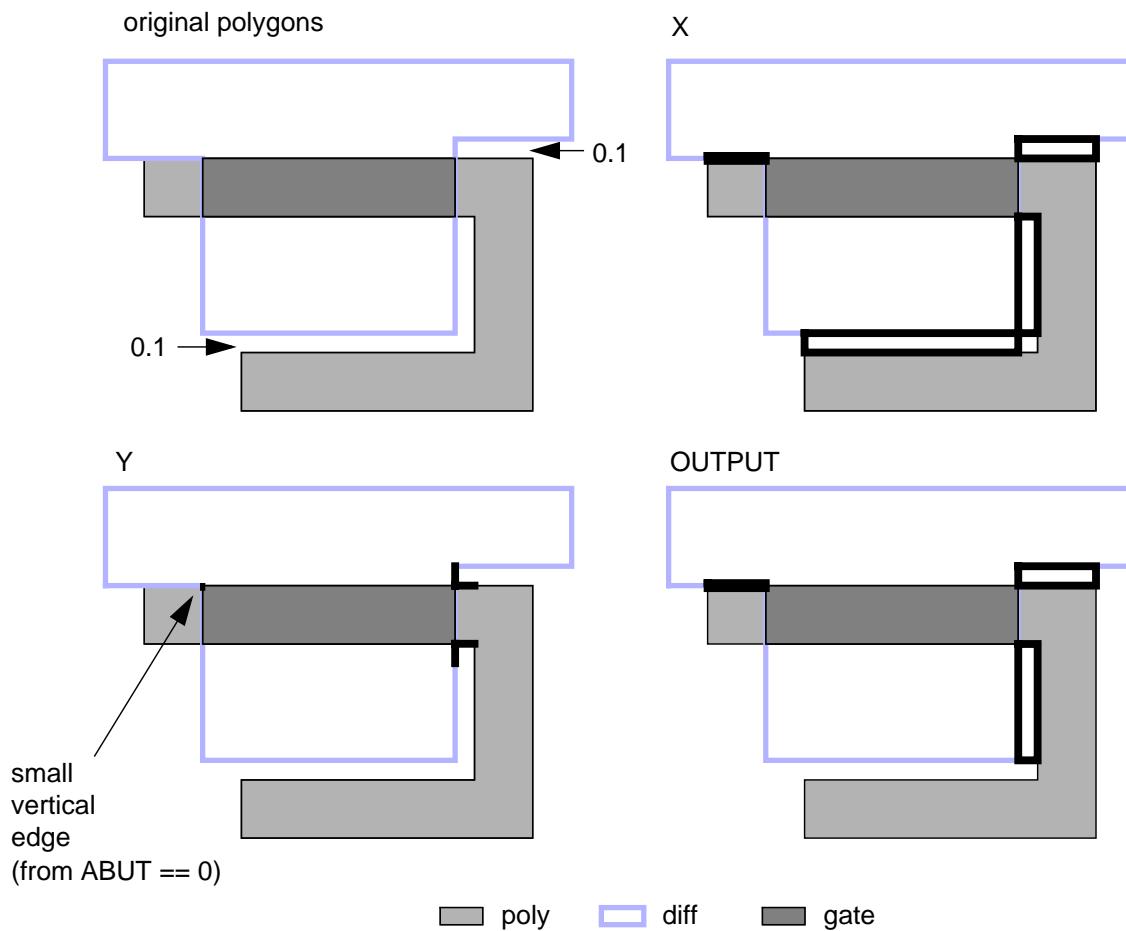
### Poly-diffusion spacing at gate bend

```

poly_to_diff1 {
  @ Normally, there is no spacing rule between poly and
  @ diffusion. However, when either poly or diffusion bends
  @ after forming a gate, the spacing must be 0.2 microns with
  @ no touching allowed.
  // potential error regions follow:
  x = EXT poly diffusion < 0.2 ABUT == 0 REGION OPPOSITE
  y = EXT [ x ] gate < 0.1 ABUT == 0 SINGULAR
  // 0.1 is arbitrary
  x WITH EDGE y// real error regions
}

```

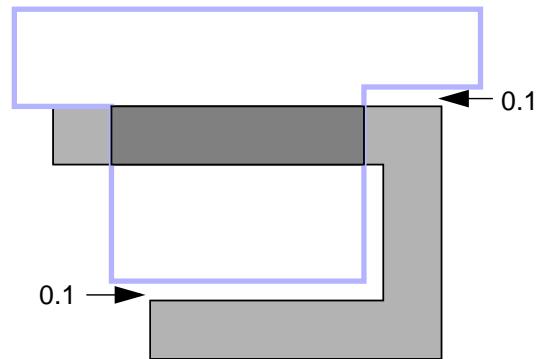
**Figure 8-11. poly\_to\_diff1**



```
poly_to_diff2 {
  @ Normally, there is no spacing rule between poly and
  @ diffusion. However, when either poly or diffusion bends
  @ after forming a gate, the spacing must be 0.2 microns with
  @ no touching allowed. Uses perpendicular only and about
  @ options:
  EXT gate diffusion < 0.2 PERPENDICULAR ONLY ABUT == 90
  EXT gate poly < 0.2 PERPENDICULAR ONLY ABUT == 90
}
```

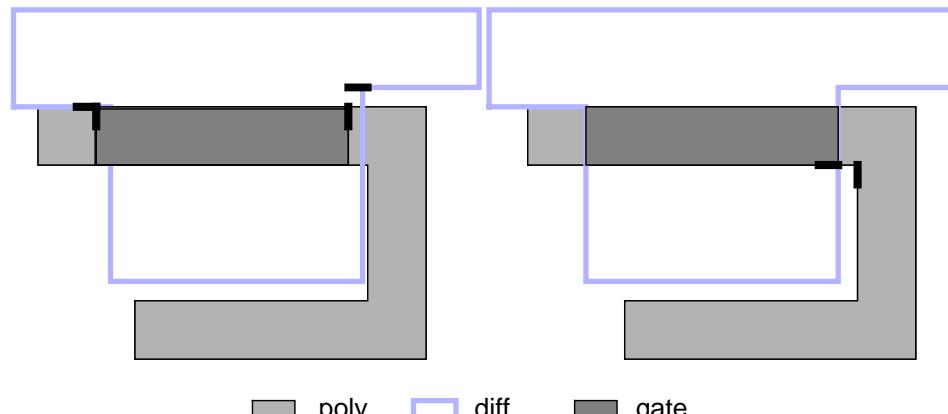
**Figure 8-12. poly\_to\_diff2**

original polygons



first output

second output

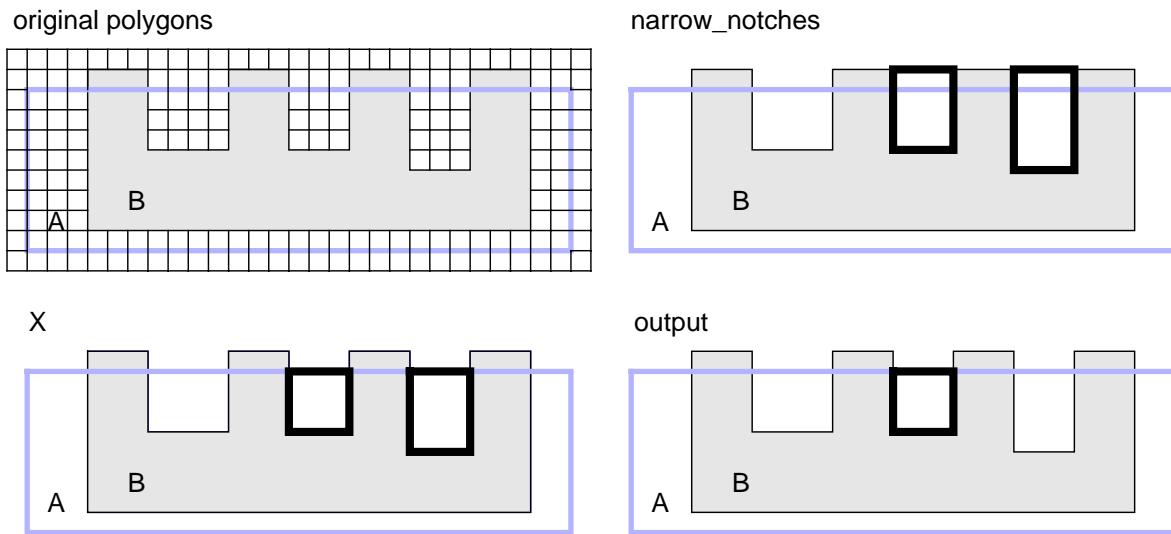


■ poly   ■ diff   ■ gate

## Notch spacing

```
NARROW_NOTCH {  
    @ If the width of a notch in mask B is less than 4, then the  
    @ area of that notch coincident with mask A must be greater  
    @ than or equal to 12.  
    narrow_notches = external B < 4 region notch  
    X = narrow_notches and A  
    area X < 12  
}
```

**Figure 8-13. NARROW\_NOTCH**



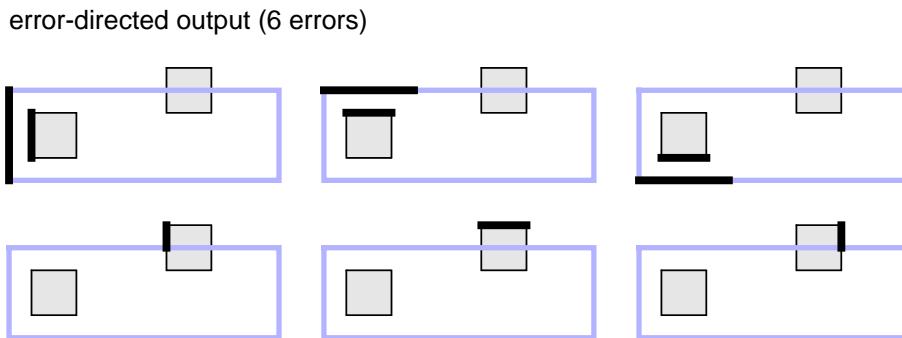
## Enclosure and Extension Checks

This section contains nmDRC enclosure and extension checks. Both categories of check are typically handled by the ENCclosure operation.

### Reduction of errors using polygon-directed output

```
Rule_53a {
ENCLOSURE cont met < 2 ABUT == 0 SINGULAR OUTSIDE ALSO
}
```

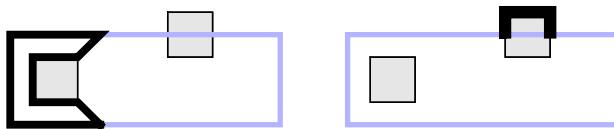
**Figure 8-14. Rule\_53a**



```
Rule_53b { // polygon-directed output
ENCLOSURE cont met < 2 ABUT == 0 SINGULAR OUTSIDE ALSO REGION }
```

**Figure 8-15. Rule\_53b**

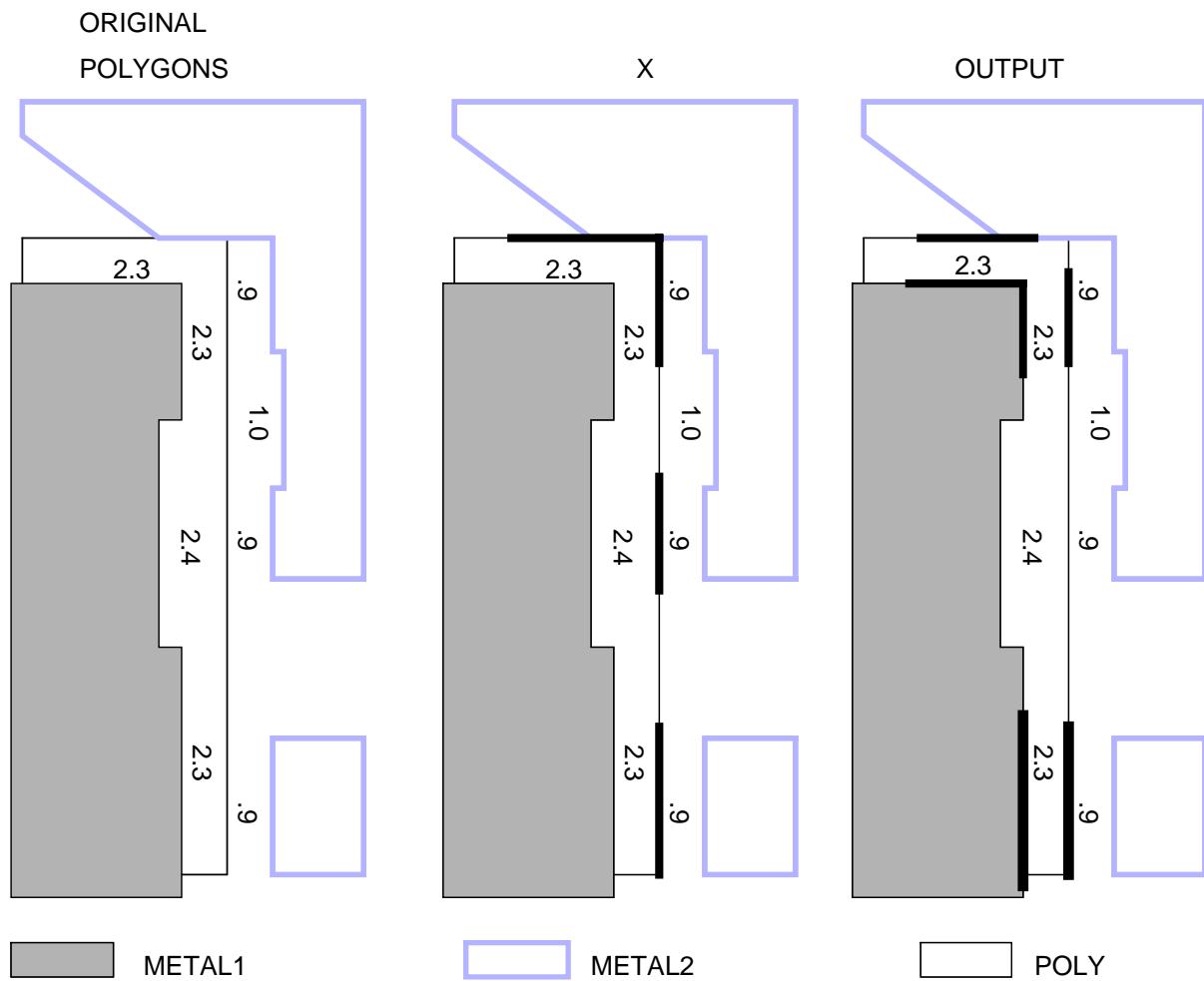
polygon-directed output (2 errors)



### Layer1 extension over layer2

```
PEM { @ POLYSILICON EXTENSION OVER METAL-1 CHECK
@ Poly extension over metall must be 2.4 microns if metal2
@ is present. Metal2 is defined to be present if the
@ spacing between metal2 and poly is less than 1 micron
X = external [poly] metal2 < 1 abut
enclosure metall X < 2.4
}
```

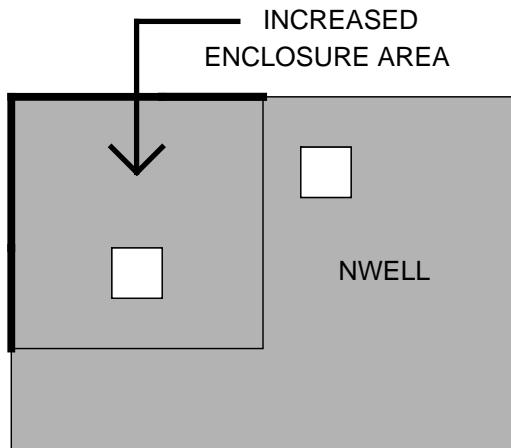
**Figure 8-16. PEM**



## Tap enclosures

```
Rule_987 {
@ Enclosure of ntaps by nwell must be 5 microns in general but
@ 6 microns whenever the enclosed ntap is within a 10 micron
@ square of a 90-degree inside corner of an nwell. Inside
@ touching is not allowed.
// First catch the normal errors.
ENC ntap nwell < 5 ABUT == 0 SINGULAR
// The following finds all edges at 90-degree nwell corners
// and trims them to within 10 microns of the corner. The
// intersecting only filter makes sure that incidental nwell
// widths < 10 microns are not selected.
x = INT [nwell] < 10 ABUT == 90 INTERSECTING ONLY
// The following checks the exception clause of the rule. The
// opposite metric assures that the ntap actually intersects
// the 10 micron square.
ENC ntap x >= 5 < 6 OPPOSITE
}
```

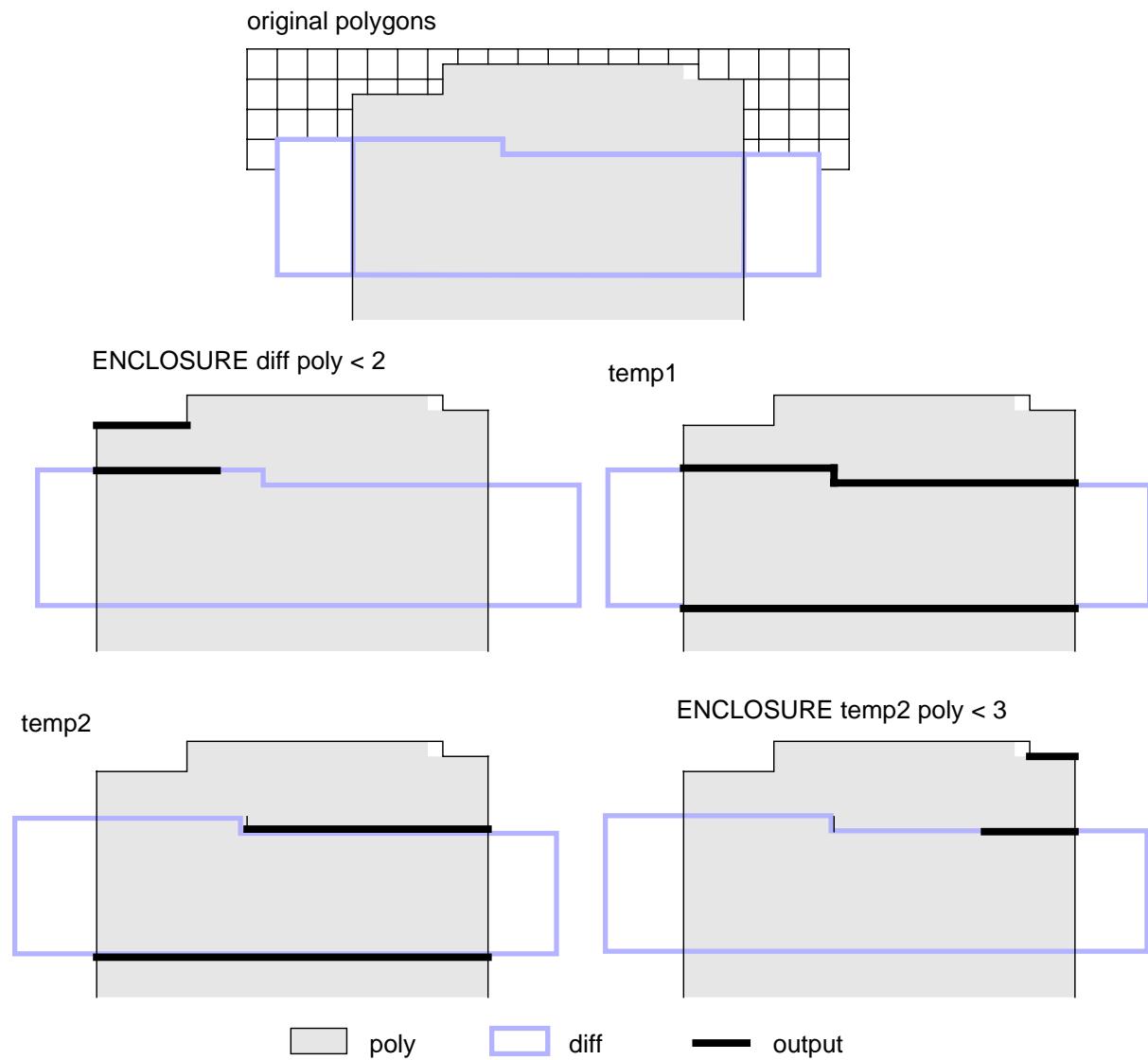
**Figure 8-17. Rule\_987**



## Poly-diffusion extension

```
// POLYSILICON EXTENSION OVER TRANSISTOR GATE RULE
// Polysilicon extension over transistor gate must be 2
// microns in general and 3 microns whenever the enclosed
// diffusion edge length is greater than 8 microns.
POLY_EXTENSION_ERROR {
temp1 = diff inside edge poly
temp2 = length temp1 > 8
enclosure diff poly < 2
enclosure temp2 poly < 3
}
```

**Figure 8-18. POLY\_EXTENSION\_ERROR**



## Extension with enclosure

```
ABC {  
@ Polygons of B which overlap polygons of A must extend past A  
@ 2 microns unless C encloses this extension and extends past  
@ A 4 microns; in this case B need only extend past A by 1  
@ micron. C must touch or overlap A to be considered but B  
@ touching A is not considered.  
W = A not outside edge C  
X = A outside edge C  
Y = enclosure [W] C < 4 MEASURE COIN  
Z = enclosure (W) C < 4 MEASURE COIN  
enclosure X B < 2  
enclosure Y B < 2  
enclosure Z B < 1  
}
```

Figure 8-19. ABC Check (part 1)

original polygons

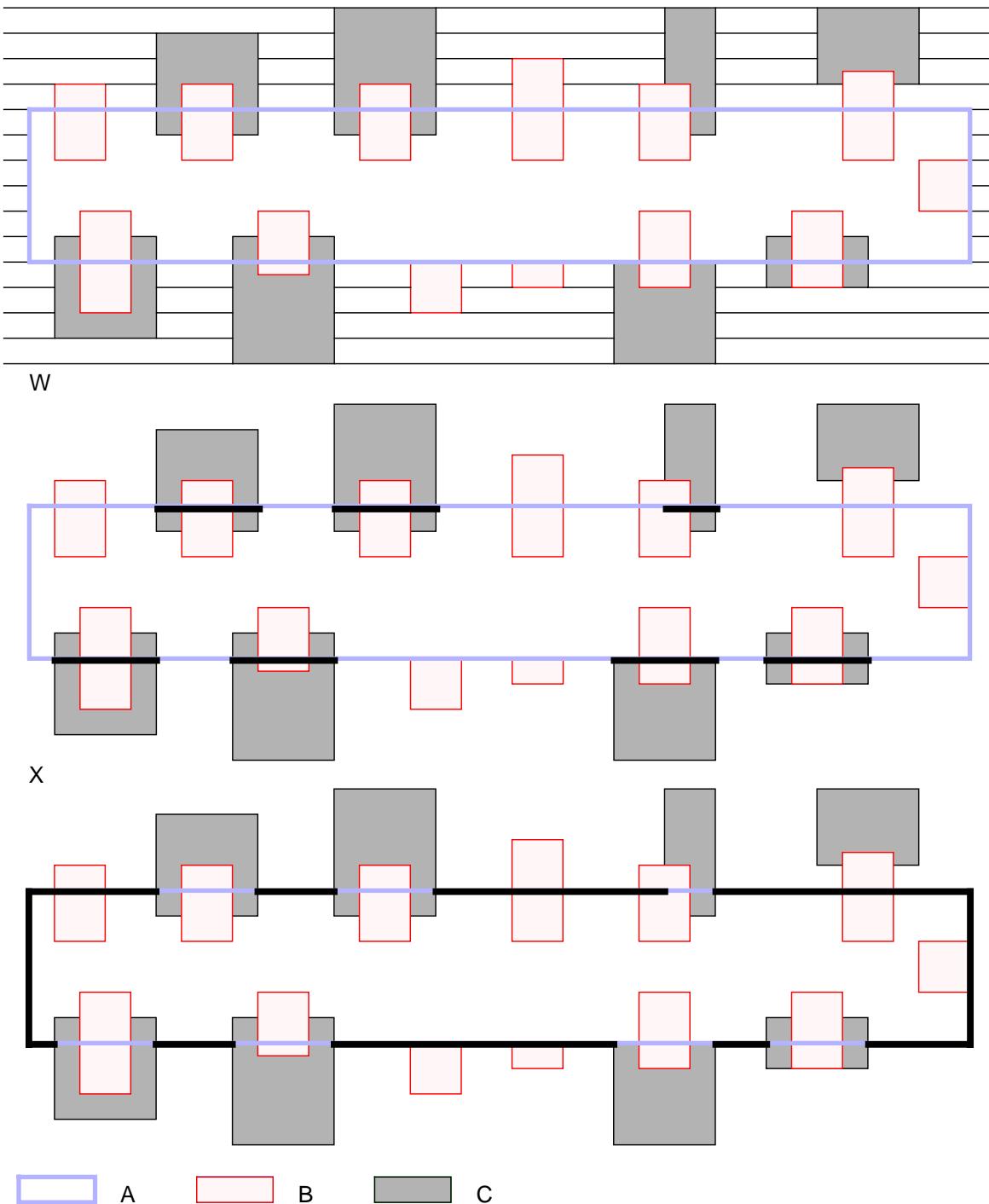
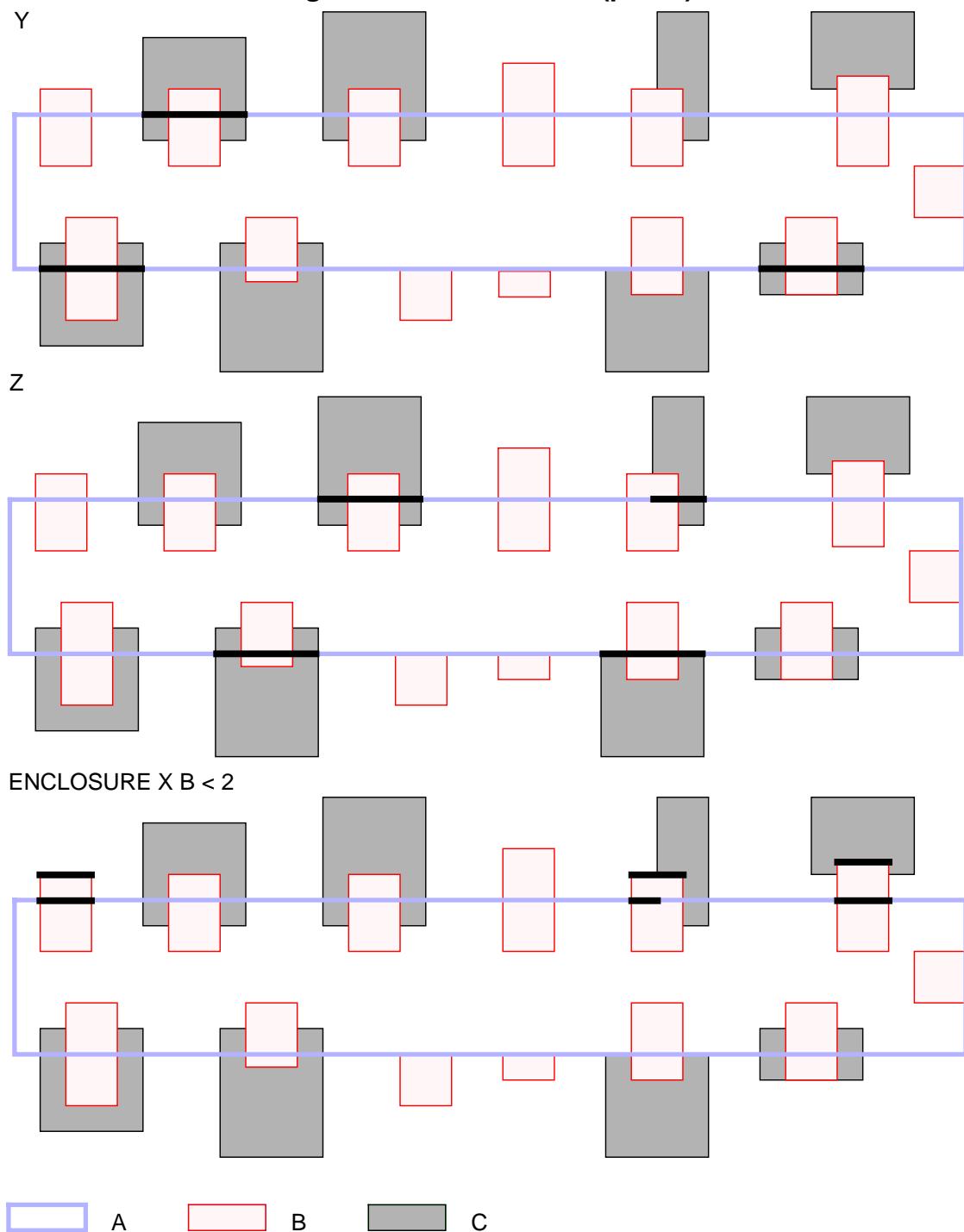
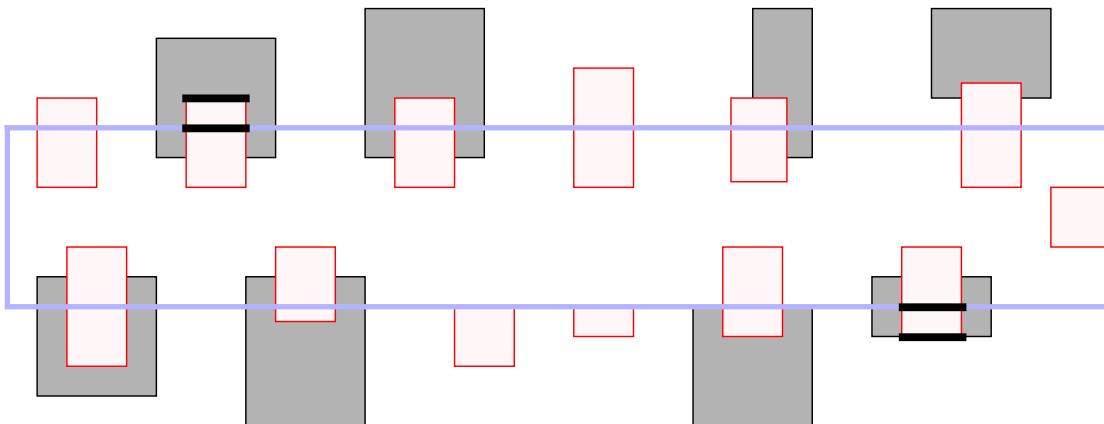


Figure 8-20. ABC Check (part 2)

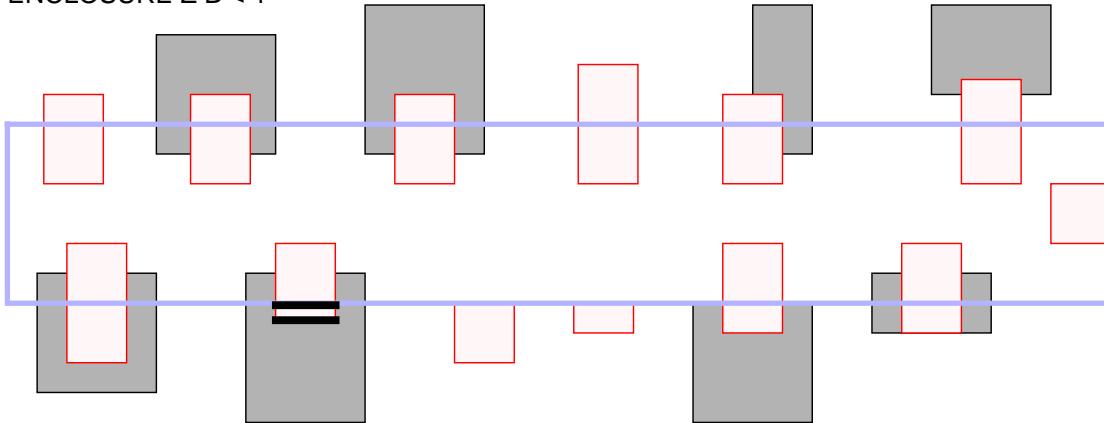


**Figure 8-21. ABC Check (part 3)**

ENCLOSURE Y B < 2



ENCLOSURE Z B < 1



[A] [B] [C]

## Contact Checks

This section contains examples of nmDRC contact layer checks.

### Contacts must be exactly X by Y

```
Contact_Size { @ Exact: 2 X 3 um
NOT RECTANGLE contact == 2 BY == 3
}
```

### Minimum via spacing

```
bad_via { @ via less than minimum via spacing
    x = SIZE via BY min_via_space OVERLAP ONLY
    via cut x
}
```

## Isolated contact spacing

```
iso_contact { @ find isolated contacts outside maximum spacing
    NOT WITH NEIGHBOR cont >= 1 SPACE <= max_cont_space
}
```

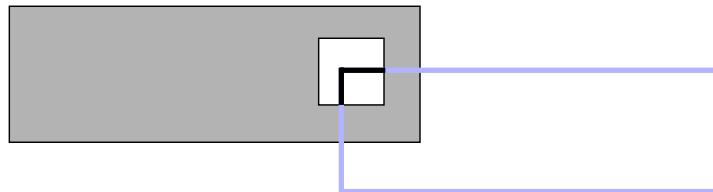
## Contacts having coincident edge with metal

```
// Select all contacts which are inside metal and have a
// coincident edge with metal.
contact_touch_inner_metal {
    x = contact inside metal
    y = x coincident inside edge metal
    x interact y
}
```

## Contacts with incomplete overlap

```
// Show contacts with incomplete overlap.
NON_OVLP_CNT {
    x = metal INSIDE EDGE contact
    contact interact x
}
```

**Figure 8-22. NON\_OVLP\_CNT**

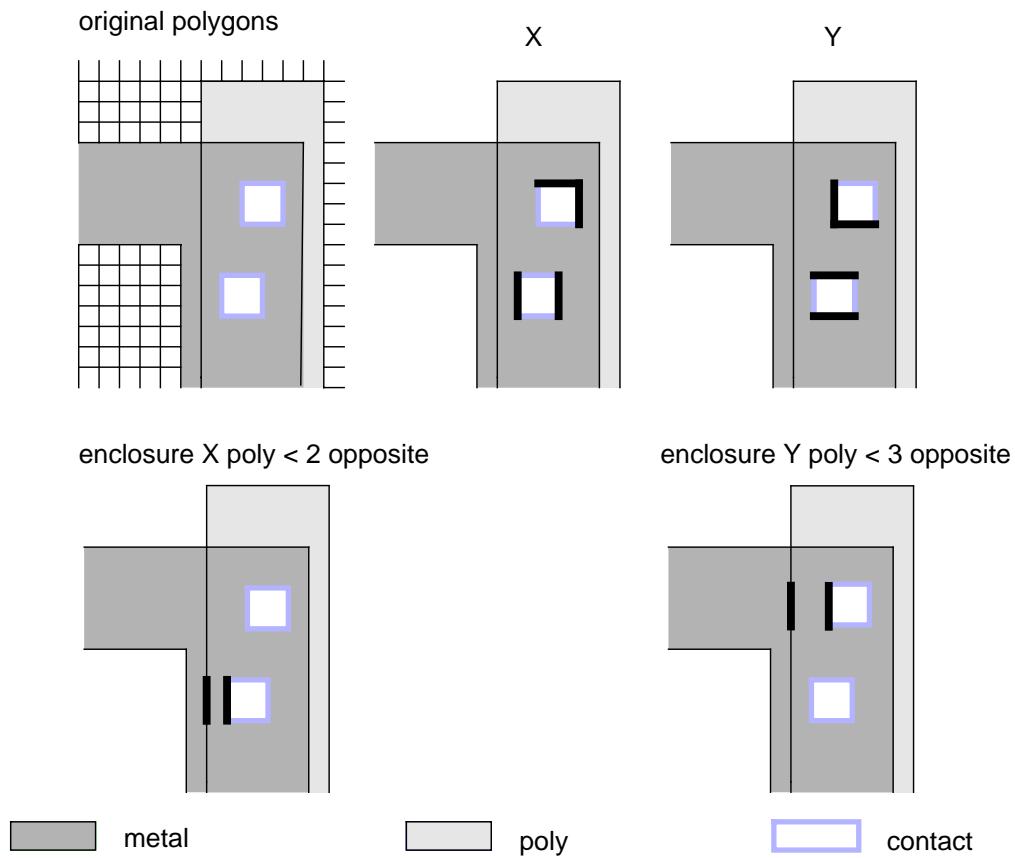


## Spacing with current flow direction dependencies

```
I_DIR_SPACING {
    @ Enclosure of metal contact by poly must be greater than or
    @ equal to 3 microns in the direction of current flow in
    @ metal; otherwise the enclosure must be greater than or equal
    @ to 2 microns. Current flow direction in metal is defined as
    @ any metal direction where metal surrounds contact by more
    @ than 3 microns.

    // X = edges not in current flow direction
    // Y = edges in current flow direction
    X = ENCLOSURE [ contact ] <= 3 metal OPPOSITE
    Y = ENCLOSURE ( contact ) <= 3 metal OPPOSITE
    ENCLOSURE X poly < 2 OPPOSITE
    ENCLOSURE Y poly < 3 OPPOSITE
}
```

**Figure 8-23. I\_DIR\_SPACING**



### Well tie-down contact rule

```
WTDC {@ WELL TIE-DOWN CONTACT RULE
@ p+ contacts within well may be 4 microns from well edges as
@ long as p+ diffusion covers the area from the contact to the
@ well edge. Otherwise, all contacts must be 6 microns from
@ well edges.

// The first four statements generate the regions of interest.
// contact_in_well and pp_in_well contain those polygons from
// contact layer and p+ layer, respectively, which are inside
// well.contact_in_well is further partitioned into those
// contacts or portions thereof which are inside pp_in_well
// (c_in) and those contacts or portions thereof which are
// inside well but not inside p+ (c_not_in).
contact_in_well = contact inside well
pp_in_well = pp inside well
c_in = contact_in_well and pp_in_well
c_not_in = contact_in_well not pp_in_well
// We can check contacts in c_not_in immediately for an
// enclosure violation of 6 microns against well edges and
// c_in immediately for an enclosure violation of 4 microns
// against well edges.
enclosure c_not_in well < 6 abut == 0 opposite
enclosure c_in well < 4 abut == 0 opposite
```

```
// The next operations select c_in edges and well edges for
// which enclosure of c_in by well is between 4 and 6 microns.
// These edges will go into layers C1 and Y, respectively.
C1 = enclosure [ c_in ] well >= 4 < 6 opposite
Y = enclosure c_in [ well ] >= 4 < 6 opposite
// Now comes the interesting part. We wish to determine
// whether p+ diffusion actually covered the area between C1
// edges and well. We first subtract the Y edges from p+.
// Next, C1 is measured against the remainder of the p+ edges
// for an enclosure violation of 6 microns. This will
// determine whether any p+ edges blocked the area between the
// C1 edges and the Y edges. The use of the opposite
// metric here and above will eliminate bringing ends of the
// remaining p+ edges into the measurement process.
Z = pp_in_well not coincident edge Y
enclosure [ C1 ] Z < 6 abut == 0 opposite
}
```

**Figure 8-24. WTDC Polygons**

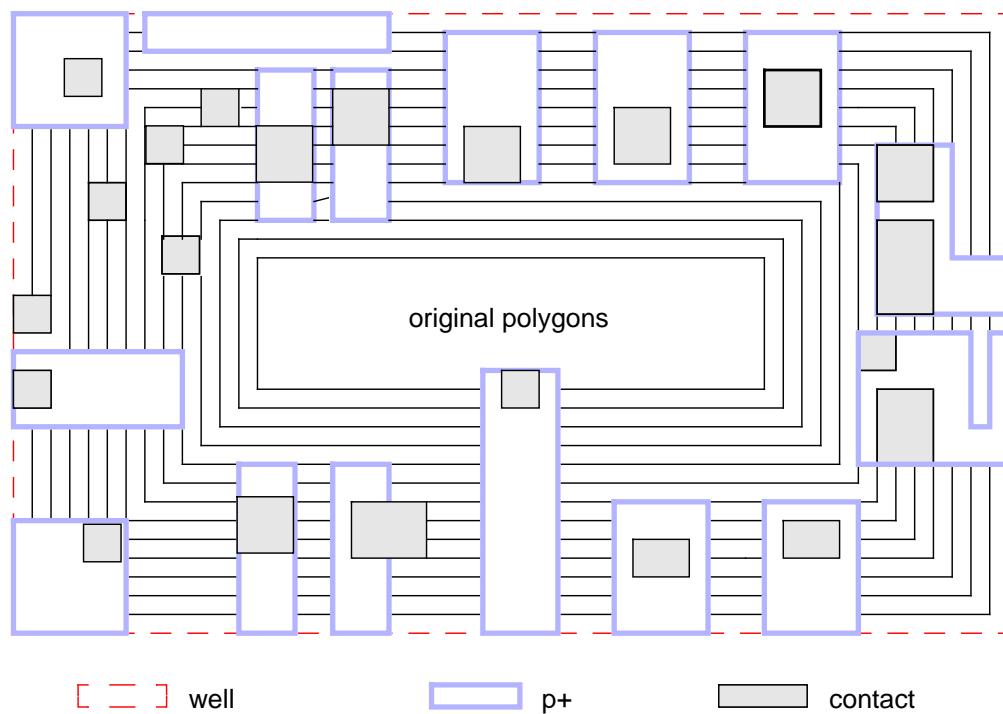


Figure 8-25. WTDC C\_IN

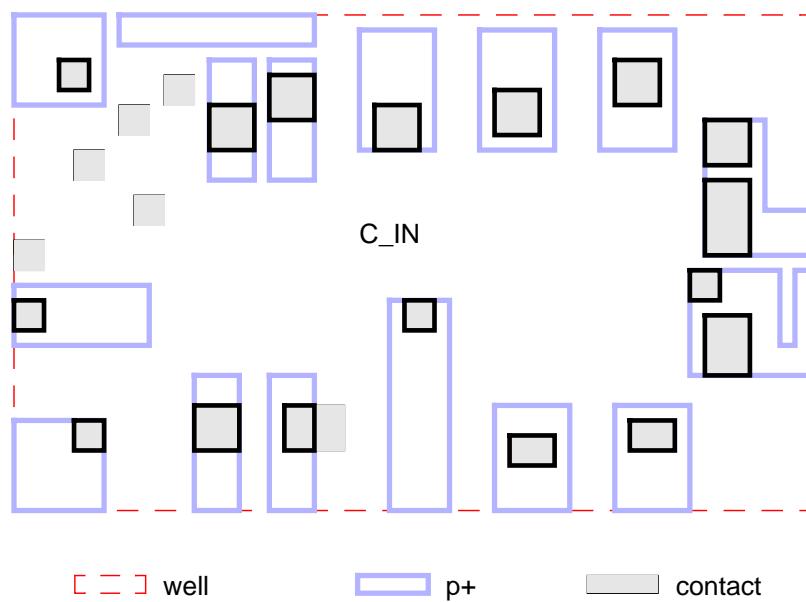
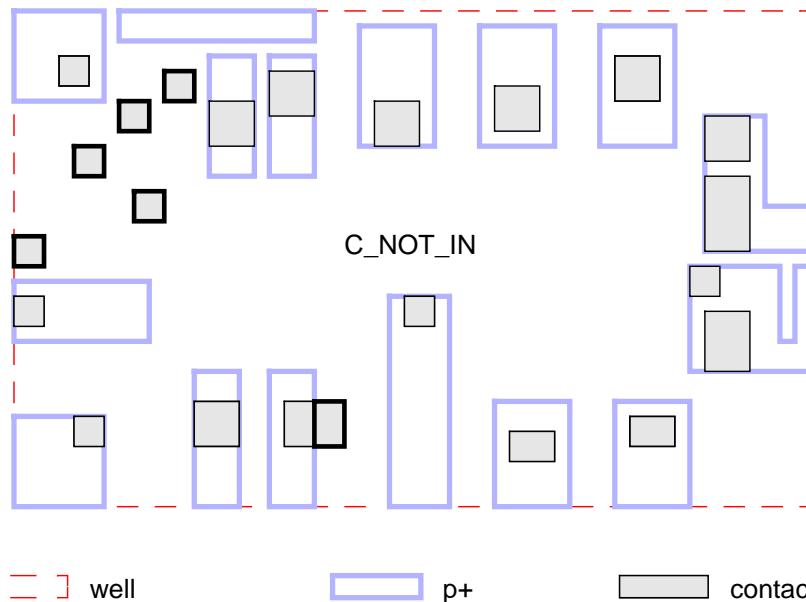
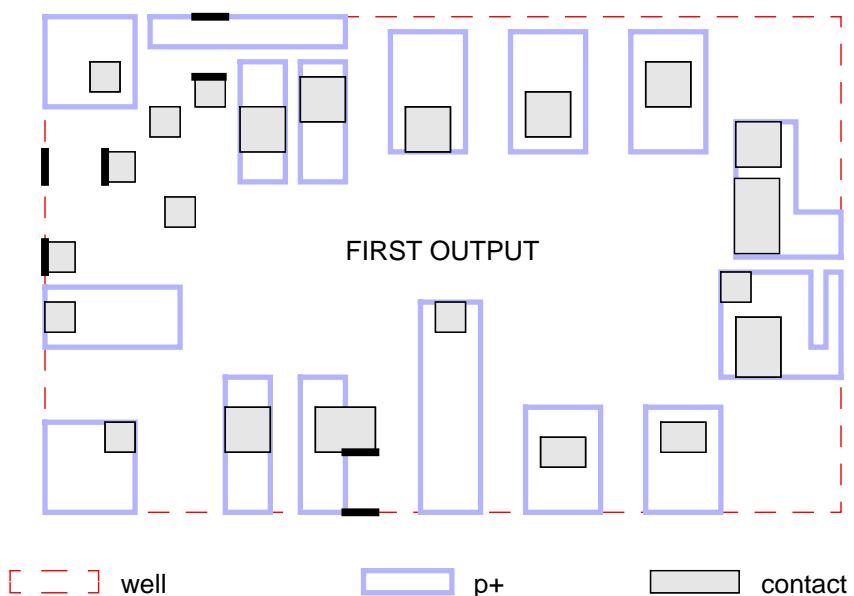


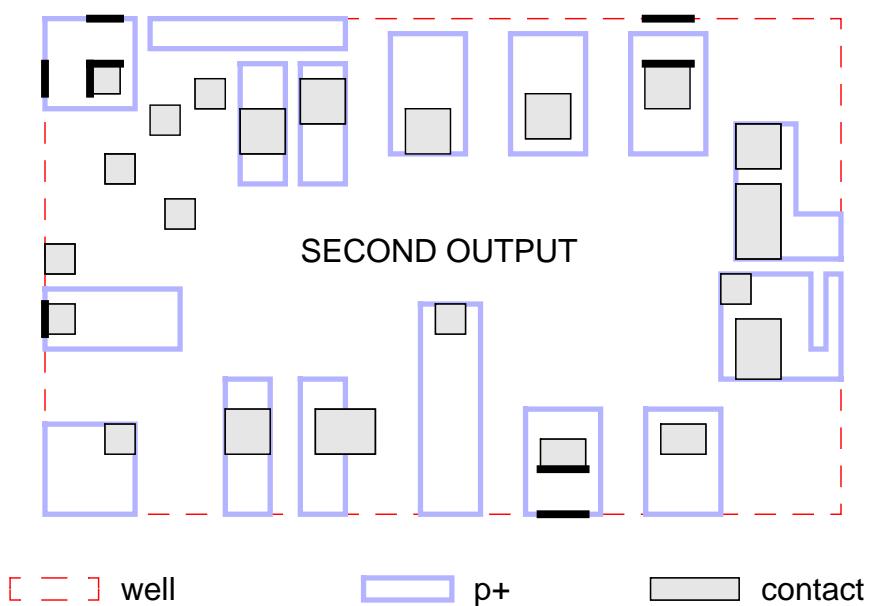
Figure 8-26. WTDC C\_NOT\_IN



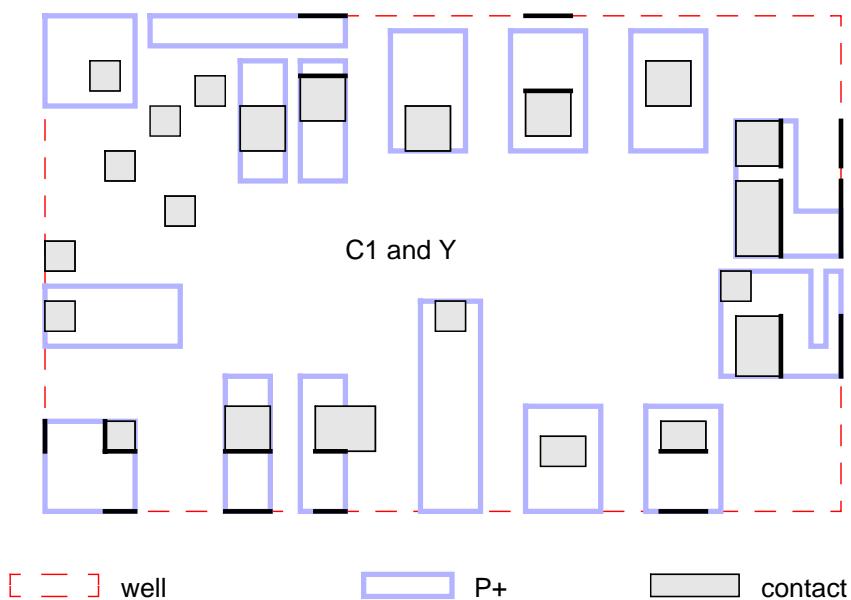
**Figure 8-27. WTDC First Output**



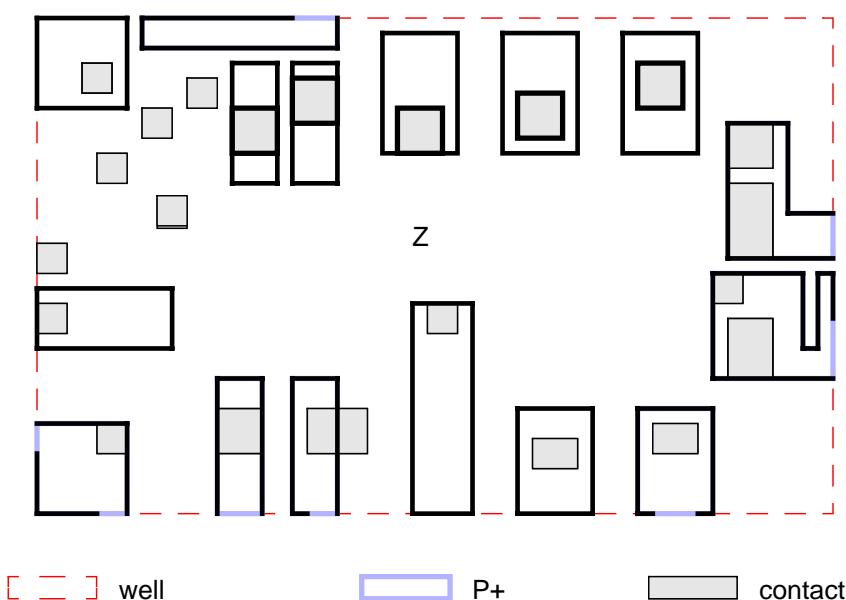
**Figure 8-28. WTDC Second Output**



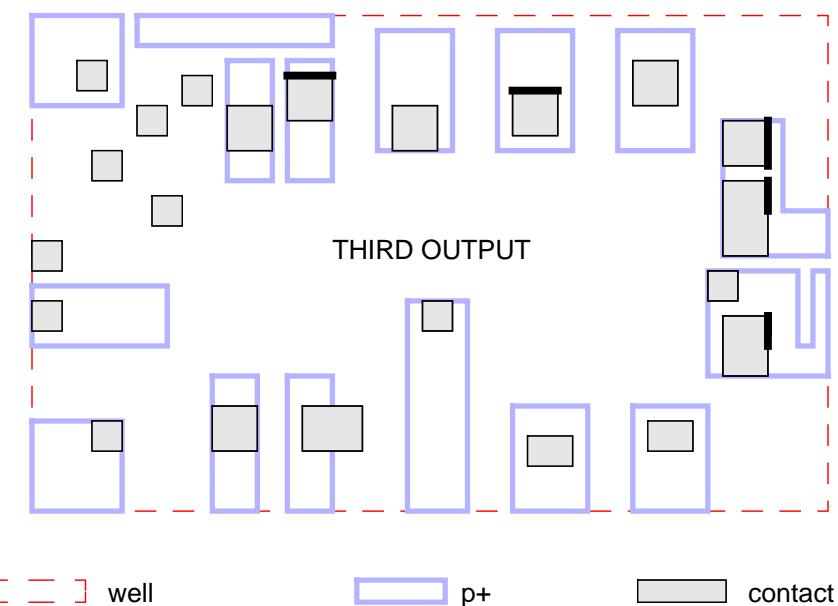
**Figure 8-29. WTDC C1 and Y**



**Figure 8-30. WTDC Z**



**Figure 8-31. WTDC Third Output**



### Check for singular touching

```
// Contact enclosure by poly = 0.6,crossing edges not ok,
// coincident inside edges not ok,
// point-to-point touching not ok:
rule_NW1 {
EXT NWELL NDIFF < 3 ABUT == 0      OVERLAP      SINGULAR
}
```

### Minimum overlap of metal over contact

```
Metal_Overlap_CT { @ Minimum: ^metal_overlap_ct um
ENCLOSURE contact metal < metal_overlap_ct
ABUT == 0 SINGULAR OUTSIDE ALSO REGION
// output as polygons
}
// You typically also want to flag contacts which are
// floating completely free, hence the OUTSIDE ALSO.
```

### Contact area within given range

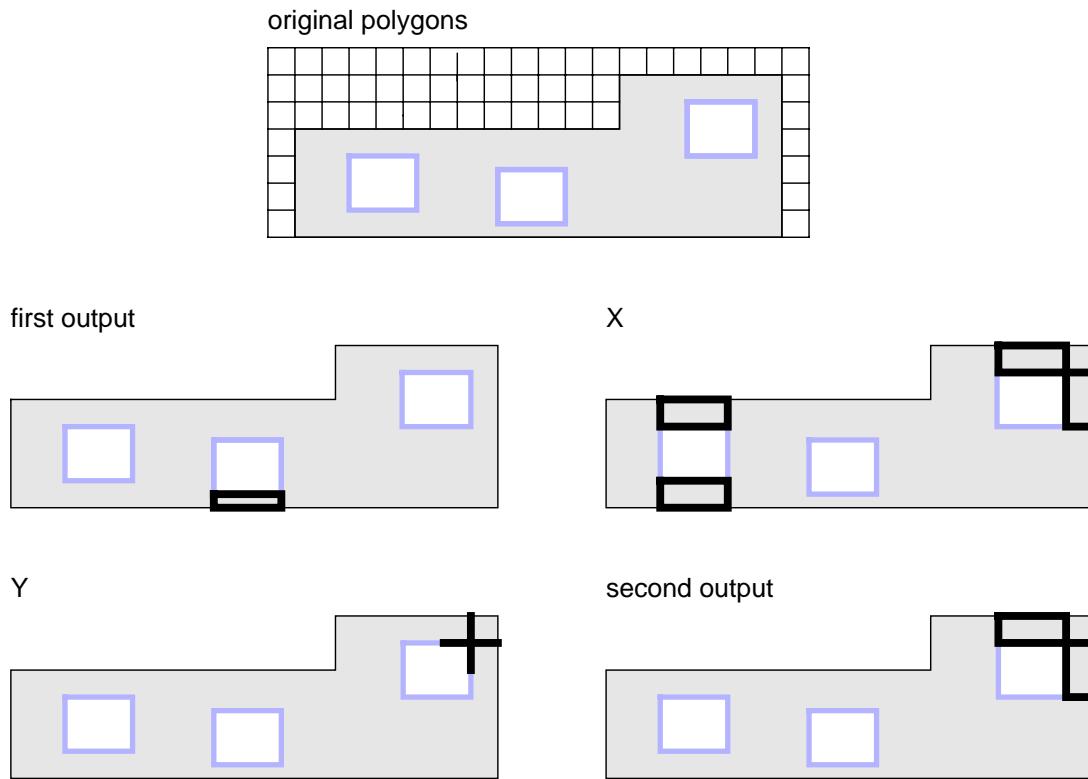
```
// Check that area of contacts is greater than 3.2 square
// microns but not greater than 5.6 square microns.
contact_area_check { not area contact > 3.2 <= 5.6 }
```

### Metal over contact with different x and y constraints

```
metal_over_contact {
@ Metal must enclose contact by 1 micron.
@ However, enclosure by 1 micron is not allowed on adjacent
@ sides of a contact. Here the enclosure must be
@ increased to 2 microns on at least one side.
```

```
// Normal rule follows:  
ENC contact metal < 1 ABUT == 0 OUTSIDE ALSO SINGULAR REGION  
// potential error regions:  
x = ENC contact metal >= 1 < 2 REGION OPPOSITE  
y = EXT [ x ] < 1 SINGULAR INTERSECTING ONLY// 1 is arbitrary  
// real error regions:  
x WITH EDGE Y  
}
```

**Figure 8-32. metal\_over\_contact**



## Contact-via separation

```
// Contacts and vias must be separated by 3 microns and no  
// stacking or touching is allowed. Without the inside also  
// option, this check would require two operations for complete accuracy:  
cont_via1{  
EXTERNAL contact via < 3 ABUT == 0 SINGULAR  
// Spacing and touching.  
AND contact via // Common area (stacking).  
}  
  
// Using the inside also option eliminates the boolean operation:  
cont_via2{  
EXTERNAL contact via < 3 ABUT == 0 SINGULAR INSIDE ALSO  
}
```

# Deriving Layers

This section contains examples of layer derivation.

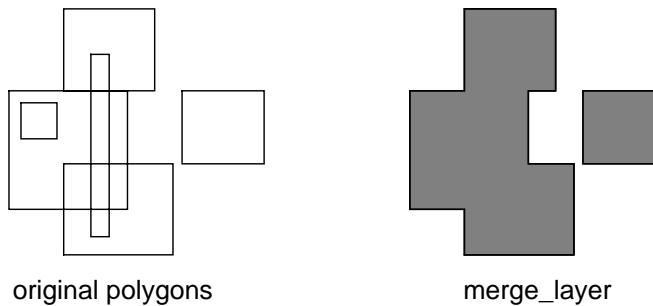
## Intersection of polygons on one layer

```
// Checks for intersection of metal blockages.  
metal_blockage_error { and metal_blockage }  
  
// Checks for intersection of sub-cell perimeters.  
// Be careful not to include the top-cell perimeter.  
perimeter_error { and metal_perimeter > 2 }
```

## Derive layer from merging overlapping polygons

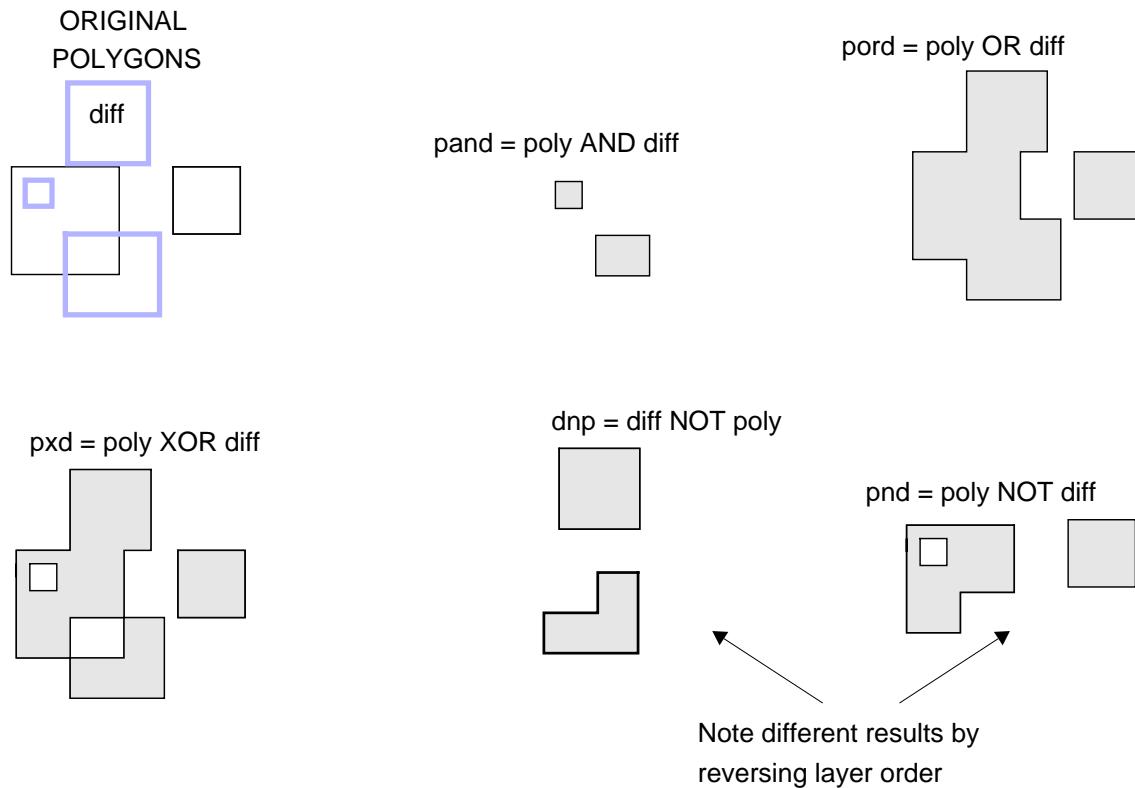
```
merge_layer = OR layerx
```

Figure 8-33. merge\_poly



## Derive layer from intersecting polygons in two layers

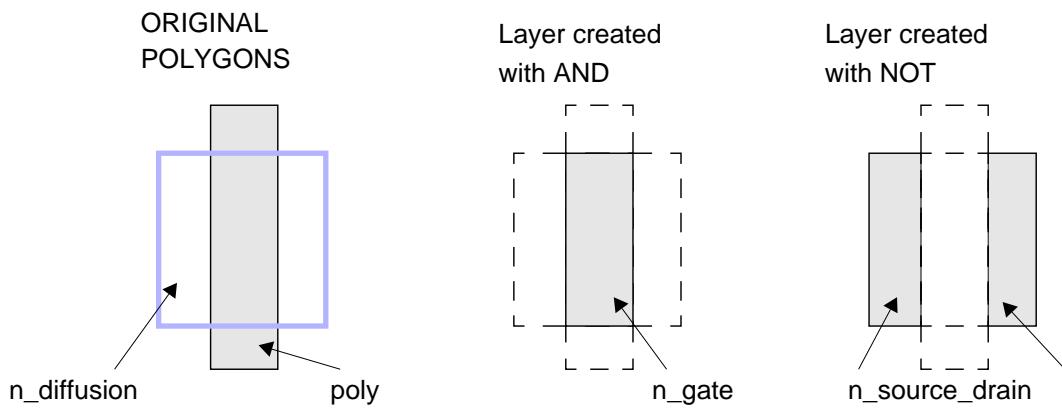
**Figure 8-34. Layer Derivations**



## Derive layers that isolate source/drain regions

```
n_gate = and poly n_diffusion
n_source_drain = n_diffusion not n_gate
```

**Figure 8-35. Source/Drain Regions**



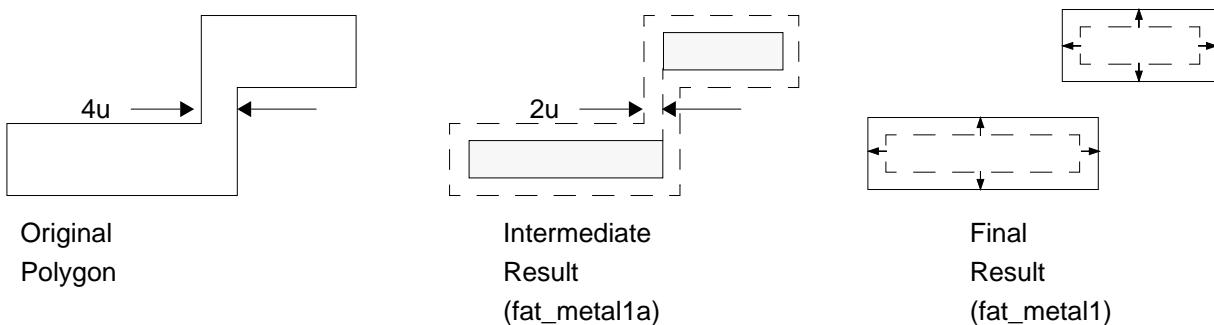
## Derive layer with objects with limited vertices

```
// Generate a clone of the metal layer with all polygons
// having less than 1024 vertices.
not_lots_of_metal_vertices = vertex <= 1024 metal
```

## Derive layer of wide objects

```
// Find metal 1 regions where the width is greater than 4
// microns:
fat_metal1a = SIZE metal1 BY -2 // Width <= 4 goes away.
fat_metal1 = SIZE fat_metal1a BY 2
// Restore metal that didn't go away.
```

**Figure 8-36. Find Wide Metal**



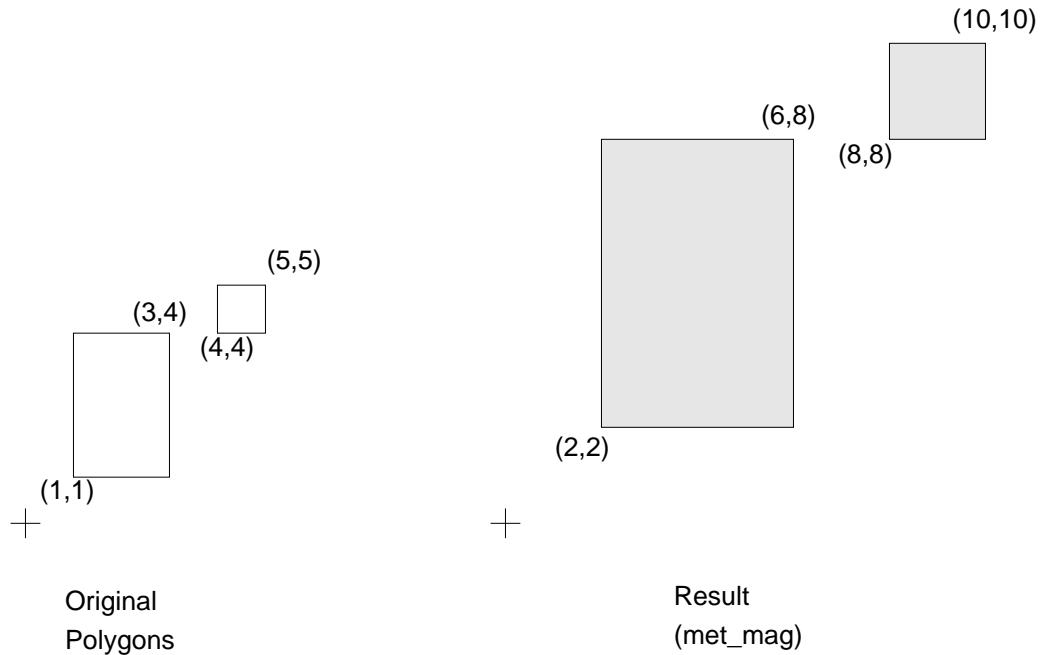
## Derive a layer by shifting a source layer

```
// create a layer called met_shift that is a copy of metal,
// that is shifted right (x) by two units, and down (y) by 1.5
// units.
met_shift = SHIFT metal BY 2    -1.5
```

## Derive an enlarged or reduced scale layer

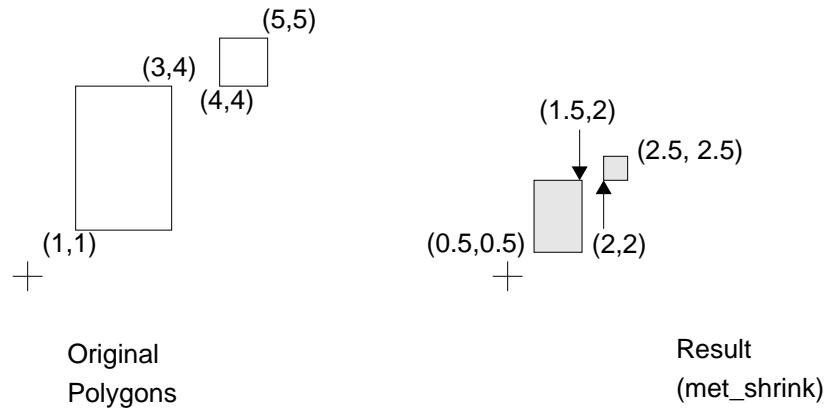
```
// create a layer called met_mag that is an enlarged copy of
// metal, whose objects' x and y coordinates are multiplied by
// 2 (a positive floating point value).
met_mag = MAGNIFY metal BY 2
```

**Figure 8-37. Using Magnify to Enlarge a Layer**



```
// create a layer called met_shrink that is a reduced copy of
// metal, whose objects' x and y coordinates are multiplied by
// 0.5 (a positive floating point value).
met_shrink = MAGNIFY metal BY 0.5
```

**Figure 8-38. Using Magnify to Shrink a Layer**



### Derive layer of gate edges along poly

```
// Select gate edges along poly.
// Note that the gate layer is not explicitly generated.
poly_gate_edges = poly INSIDE EDGE diff
```

## Other nmDRC Checks

This section contains miscellaneous nmDRC checks.

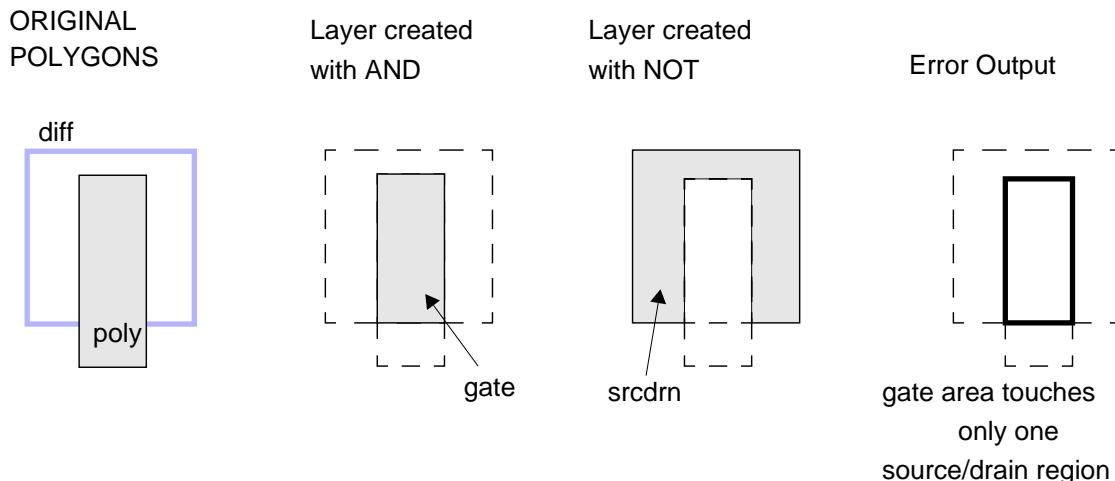
### Derive transistor regions

```
n_diff = diffusion NOT p_dope // n+ diffusion
p_diff = diffusion AND p_dope // p+ diffusion
n_tap = n_diff NOT OUTSIDE n_well // n-tap areas
not_n_tap = n_diff OUTSIDE n_well
// Areas which are not n-taps
p_tap = p_diff OUTSIDE n_well // p-tap areas
not_p_tap = p_diff NOT OUTSIDE n_well
// Areas which are not p-taps
n_gate = poly AND not_n_tap // n-channel gates
p_gate = poly AND not_p_tap // p-channel gates
nsd = not_n_tap NOT n_gate // n-source/drain regions
psd = not_p_tap NOT p_gate // p-source/drain regions
```

### Gates that do not completely cross diffusion

```
bad_gates {
@ "Gates" where poly does not completely cross diffusion.
gate = poly AND diff // Gate regions.
srcdrn = diff NOT poly // Source/drain regions.
gate NOT TOUCH srcdrn == 2 // Must touch two src/drн regions.
}
```

**Figure 8-39. bad\_gates**



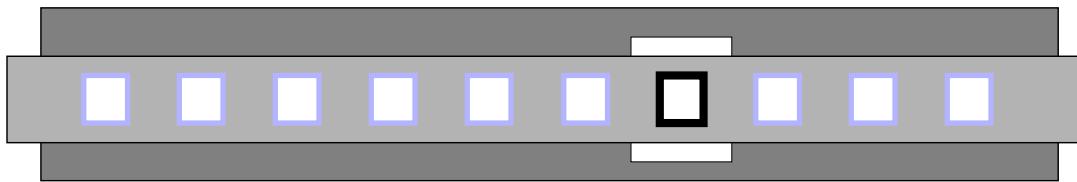
### Objects in holes of another layer

```
contacts_in_holes {
@ Select all contacts which are exactly in a hole of
@ diffusion:
X = holes diff // First find diffusion holes
```

```
Y = X inside contact
// Next pick contacts coincident with the holes
contact inside Y
}
```

**Figure 8-40. contacts\_in\_holes**

Error Output



## Holes in a layer

```
closed_metal {
@ donut structures not allowed on metal layer
    metal donut
}
```

## Wide metal that is not power or ground

```
CONNECT metal
FAT_METAL_NOT_POWER {
vdd_metal = metal net vdd
vss_metal = metal net vss
power = vdd_metal OR vss_metal
wide_met = SIZE metal BY -2// Width <= 4 goes away.
fat_metal = SIZE wide_met BY 2
// Restore metal which did not disappear
fat_metal NOT power
//generate error output when fat metal is not vdd or vss
}
```

## Polygons with an edge longer than 100 microns

```
// Find all metal polygons with at least one edge longer than 100 microns.
long_metal {
x = metal length > 100
long_metal_polygons = metal interact x
}
```

## Acute angles, off-grid vertices, and skewed edges

```
flag_check {
@ Find acute angles, off-grid vertices, and edges which are
@ non-orthogonal or non-45
    DRAWN ACUTE
    DRAWN OFFGRID
    DRAWN SKEW
}
```

## Sub-cell perimeters inside that of the top-level cell

```
// ICrules only.
// Check that no sub-cell perimeter is outside that of the
// top-level cell:
perim_check {
    // Cell perimeters are stored on a special layer called
    // "perimeter" as objects having both internal and external
    // aspect.
    X = TOPEX perimeter// The perimeter of the top-level template
    perimeter NOT X
}
```

## Gate alignment

```
non_aligned_gate {
    @ Gates must be horizontal along the edge inside poly and
    @ vertical along the edge inside diffusion. Flag gates
    @ themselves not gate edges.
    gate = poly and diff
    gate_edge_in_poly = gate coincident edge diff
    gate_edge_in_diff = gate not coincident edge diff
    bad_angle_1 = gate_edge_in_poly not angle == 0
    bad_angle_2 = gate_edge_in_diff not angle == 90
    bad_gate_1 = gate with edge bad_angle_1
    bad_gate_2 = gate with edge bad_angle_2
    bad_gate_1 or bad_gate_2// Kill duplicate errors.
}
```

## Length to width ratio

```
// check length-to-width ratio of chip
chip_too_long_or_wide {
    chip = EXTENT
    NOT RECTANGLE chip > min < max ASPECT 1 // chip must be square
}
```

## Pad metal to unrelated diffusion check

This will check pad metal to unrelated diffusion. Related diffusion is diffusion tied to pad metal. Unrelated diffusion is everything else.

```
//diff in this case is all diff you want to check.
//make copies of your present metals and vias so you
//don't mess up your connectivity.
cm1 = COPY met1
cm2 = COPY met2
cv = COPY via
CONNECT cm2 pad
CONNECT cm2 cm1 by cv
//In other words, all cm2 on the same net as pad
m2x = NET AREA RATIO cm2 pad > 0
m1x = NET AREA RATIO cm1 pad > 0
cx = cont INSIDE m1x //all contacts in m1x
//This is the unrelated diffusion.
checkd = diff OUTSIDE cx
```

```
pad2unrelated_diff_rule {
    EXTERNAL checkd m2x < 5
    EXTERNAL checkd m1x < 5
}
```

## Missing and conflicting Stamp

The missing connections Stamp means a polygon was found which did not get a Stamp by the naa or paa respectively. This may or may not be a problem depending on the polygon. The conflicting connection Stamp means the respective well (n or p) did not receive node numbers because different nodes are trying to Stamp the same object. This could very likely be a problem and you will want to find out why you have different nodes trying to Stamp the same object.

```
ywelln = nwell NOT cwelln
//identifies missing "n" connections
miss_n_conn { ywelln OUTSIDE nq }
//identifies confl. "n" conn.
conflict_n_conn { ywelln NOT OUTSIDE nq }

ywellp = pwell NOT cwellp
//identifies missing "p" connections
miss_p_conn { ywellp OUTSIDE pr }
//identifies confl. "p" conn.
conflict_p_conn { ywellp NOT OUTSIDE pr }
```

## Using Stamp to find floating gates

```
// This rule uses STAMP to find floating gates; that is, gates
// not connected to the output of another transistor.
// Connectivity must be established for the layout (this can
// be done using $extract_cell_connectivity in Pyxis Layout

layer poly 1
layer diff 2
layer metal 3
layer cont 4

connect diff metal by cont
// Above statement forces connectivity between diff and metal
// through a contact

floating_gate {
    gate = poly AND diff
    // above creates a gate layer with connectivity info of poly

metalcont = metal AND cont
// This step isolates metal contacts.
// The next step will establish coincidence of poly and metal
// contacts.Theoretically, you are only concerned with poly
// connected to metal, but isolating the metal contact insures
// you get only intended metal to poly connections rather than
// all intersections of metal and poly

connected_poly = STAMP poly by metalcont
```

```
// This step establishes coincidence of poly and metal
// contacts

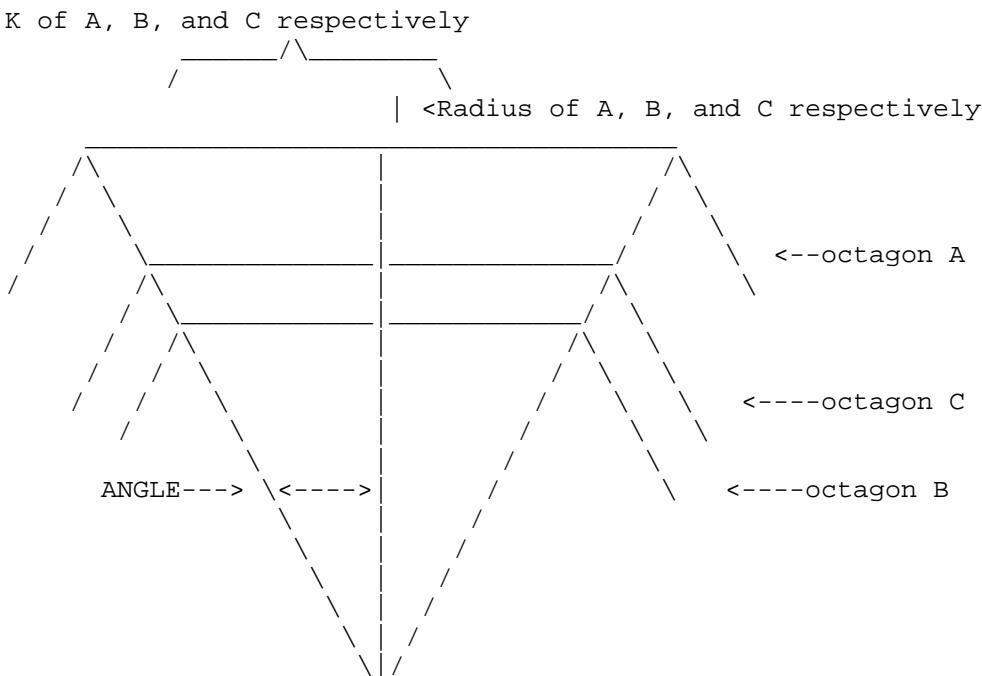
gate NOT connected_poly
// This step yields the rule check output--gate area not
// intersecting connected poly.
```

## Device Property Calculation Examples

### Device recognition

The following example is for device recognition using the built-in language for devices. This language is discussed in detail under “[Device Property Computation Built-In Language](#)” on page 1776.

This is the top of a regular octagon:



Calculations: (L = length, R = Radius, P = Perimeter)

```
Ra = Rb + L
Rc = Rb + L/3
ANGLE = 360/16 = 22.5
K/R = tangent(ANGLE)      <---BASIC FORMULA (Applies to A, B, or C)
Kb/Rb = tangent(ANGLE)
R = K/tangent(ANGLE)      <---BASIC FORMULA (Applies to A, B, or C)
Rb = Kb/tangent(ANGLE)
P = 16K                   <---BASIC FORMULA (Applies to A, B, or C)
Pb = 16Kb
Kb = Pb/16
Rb = Kb/tangent(ANGLE) = Pb/(16*tangent(ANGLE))
```

```

RC = Rb + L/3
PC = 16Kc = 16Rc*tangent(ANGLE)
Rb = Pb/16*tangent(ANGLE)
RC = Pb/16*tangent(ANGLE)+L/3
PC = 16(Pb/16*tangent(ANGLE)+L/3)*tangent(ANGLE)

```

Therefore  $W = 16(Pb/16*tangent(ANGLE)+L/3)*tangent(ANGLE)$

Property Computation:

```

DEVICE M.....
[
PROPERTY W L
Pb = (Use your derived layer of "B" to obtain perimeter here)
L = (Use your calculation to obtain length)
W = 16(Pb/(16*(tangent_ANGLE)+L/3)*tangent_ANGLE
]
NOTE: tangent_ANGLE must be provided by you.

```

## Resistor property calculation

### Example 1

```

// resistor property calculation
DEVICE R(P) ABCDE QPOLYZ QPOLYZ
[
property R
W=(perimeter_coincide(ABCDE,QPOLYZ) +
perimeter_inside(ABCDE,QPOLYZ))/2
W = W - (0.2E-6)// Adjust for real vs drawn width
L=perimeter_outside(ABCDE,QPOLYZ)/2
R = L/W*100 // 100 ohms/square
]

```

### Example 2

```

// another resistor property calculation
DEVICE r(res_ABm) ABbody2m ABtipm(pos) ABtipm(neg)
[
property res
gw = (perim_co(ABbody2m,ABtipm) +
perim_in(ABbody2m,ABtipm)) / 2
gl = (perim_out(ABbody2m,ABtipm) / 2) + 5
rbend = bends(ABbody2m) * -100
res = (gl*200 + rbend*gw)/gw
]

```

# ICVerify Applications

The following examples illustrate how to develop a rule file for ICtrace.

## Building an ICtrace Rule File

### Example 1

The following is an example of what you need for performing LVS using ICtrace. It shows a simple CMOS process with only two device types: MN and MP. It assumes that you run LVS in Pyxis Layout against the Eddm (Design Architect database).

1. Create Design Architect (DA) symbols. If you use a Mentor-supplied library, then you may already have the required DA symbols. Otherwise, you can make your own symbols as follows.
  - a. In DA, draw a MN symbol.  
Component name: mn (or anything else).  
Properties: ELEMENT = MN  
Pin names: PIN = D, G, S, B (optional)
  - b. In DA, draw a MP symbol.  
Component name: mp (or whatever you choose).  
Properties: ELEMENT = MP  
Pin names: PIN = D, G, S, B (optional)
2. Enter DA schematics. In DA, enter your schematic design using the mn and mp symbols.
3. Create EDDM viewpoint. The following is a sample DVE script for setting up an ICtrace viewpoint for a DA component called zinv.

```
> dve
open design viewpoint "zinv" "default"
add primitive "element"
check design
save design viewpoint
close design viewpoint
close session
```

4. Prepare your rule file. Below is an example of a simple ICtrace rule file for a basic CMOS process.

```
layer poly      1
layer diff      2
layer well       3
layer metal      4
layer nplus      5
```

```
layer contact      7
layer subc        11 // substrate contact

text layer poly metal

connect metal poly sd by contact
connect metal well by subc
connect metal nplus by subc

sd = diff not poly
tran = diff and poly
ptran = tran not nplus
ntran = tran and nplus

device mp ptran poly sd sd well
device mn ntran poly sd sd nplus

// END of rules file example.
```

5. Enter your layout in Pyxis. Don't exit - continue to step 6.

6. LVS. In Pyxis, bring up the ICtrace(M) palette. Execute:

**ICtrace(M) > Load Rules**

**ICtrace(M) > LVS**

The following is a sample transcript:

```
$load_rules("rules_file");
$lvs_mask(@true, @true, "zinv/default", "TEST", @eddm, "lvs.rep",
"maskdb", @true, @true);
```

### Comments

You do not need to extract a netlist for ICtrace from DA. You can run directly against the DA database, the Eddm.

You don't need to extract a device netlist from the layout. You run LVS directly on your layout in Pyxis. However, if you want to get a HSPICE or LSIM netlist from the layout for other purposes, you can do it with the WRITE MASK NETLIST command. This is on the palette:

**ICtrace(M) > Netlist**

Sample transcript:

```
$write_mask_netlist(@true, @true, @false, "", "", @eddm, "lvs.rep",
"maskdb", @true, @true, "test.net", @hspice, @simple, @layout);
```

## Example 2

This is an example of a very simple ICtrace rules file for a trivial CMOS-style process. This is not a real process but might be handy as a starting point.

```
// Trivial CMOS rules file example.

layer poly      1
layer diff      2
layer well      3
layer metal     4
layer nplus     5
layer contact   7
layer subc      11 // substrate contact

text layer poly metal

connect metal poly sd by contact
connect metal well by subc
connect metal nplus by subc

sd = diff not poly
tran = diff and poly
ptran = tran not nplus
ntran = tran and nplus

device mp ptran poly sd sd well
device mn ntran poly sd sd nplus

// END of rules file example.
```

The following info might be useful to you if you run LVS against the Eddm (Design Architect database). The list below specifies the standard schematic properties for built-in ICtrace devices. Arbitrary user defined devices are of course supported as well.

CMOS N transistor:

ELEMENT = MN  
PIN = D, G, S, B (optional)

CMOS P transistor:

ELEMENT = MP  
PIN = D, G, S, B (optional)

NMOS enhancement transistor:

ELEMENT = ME  
PIN = D, G, S, B (optional)

NMOS depletion transistor:

ELEMENT = MD  
PIN = D, G, S, B (optional)

Resistor:

ELEMENT = R  
PIN = POS, NEG

Capacitor:

ELEMENT = C  
PIN = POS, NEG

Bipolar transistor:

ELEMENT = Q  
PIN = C, B, E, S (optional)

Diode:

ELEMENT = D  
PIN = POS, NEG

If any of those properties are already in use and have different values, then other properties can be used for ICtrace. The recommended properties in that case are:

PHY\_COMP — overrides ELEMENT  
PHY\_PIN — overrides PIN

# Index

## — Symbols —

#IFDEF statements, 56

## — A —

Abbreviations, 48

Abut parameter, 523, 591, 677, 1611

Acute also parameter, 515, 584, 670

Acute only parameter, 515, 584, 670

AND operation, 115

Angle operation, 118

Angled parameter, 520, 588, 674

AREA

analyzing with respect to, 226

Area operation, 120

Attach operation, 121

## — B —

Bend determination with device property language, 1787

Boolean operations

AND, 115

NOT, 1180

OR, 1281

OR Edge, 1283

XOR, 1768

Built-in languages, 1771

## — C —

Calibre file i/o errors (Table), 2013

Calibre GDSII and OASIS output warnings (Table), 2034

Calibre GDSII/OASIS input errors (Table), 2017

Calibre GDSII/OASIS input warnings (Table), 2027

Calibre nmDRC-H warnings (Table), 2011

Calibre output warnings (Table), 2036

Calibre PERC TVF Errors (Table), 2050

Calibre precision exceptions (Table), 2013

Capacitance Order statement, 124

Cell name/location warnings (Table), 2036

Check set, 491, 492, 499, 500

Coincident Edge operation, 126

Coincident Inside Edge operation, 127

Coincident Outside Edge operation, 128

Compile-time TVF, 1891

Concurrency

for DFM Analyze, 226

for DFM Critical Area, 258

for DFM Property, 328

for DFM Transition, 420

Concurrent operations, 258

Conditionals, 56

Connect operation, 129

Connected parameter, 514, 583, 669

Connectivity extraction and stamp warnings (Table), 2038

Connectivity Extraction Errors (Table), 1964

Connectivity extraction operations

requirements and restrictions, 122, 132, 684

Connectivity operations

Net Interact, 1178

Connectivity statements

Attach, 121

Connect, 129

Disconnect, 426

Label Order, 683

Net, 1149

Net Area, 1151

Net Area Ratio, 1152

Net Area Ratio Accumulate, 1173

Not Net, 1203

Ornet, 1285

Sconnect, 1644

Stamp, 1672

Consistency checking, 1712

Consistency checking in device reduction, 1865

Constraint Errors (Table), 1938  
Constraints  
    defined, 45  
Contact checks, 1207  
Contact checks (examples), 2096  
Continuous nmDRC errors (Table), 1938  
Convex Edge operation, 134  
Copy operation, 138  
Corner parameter, 521, 589, 675  
Corner to corner parameter, 521, 589, 675  
Corner to edge parameter, 521, 589, 675  
Corresponding device definitions, 1864  
COUNT, analyzing with respect to, 226  
Cut operation, 141

— D —

Database units, 1602  
DBCLASSIFY operation, 144, 150  
Deangle operation, 154  
Density Convolve operation, 183  
Density operation, 156  
Density Operation Errors (Table), 1943  
Deriving layers (examples), 2105  
Design correction statements, 83  
Device annotation, 1842  
Device annotation built-in language, 1842  
Device Definition Errors (Table), 1945  
Device enclosure functions, 1833  
Device layer, 200  
Device Layer operation, 216  
Device Layer Operation Errors (Table), 1963  
Device operation, 196  
Device property computation built-in  
    language, 1776  
    bend determination, 1787  
    data retrieval functions, 1782  
    described, 1776  
    perimeter functions, 1819  
    well proximity enclosure functions, 1833  
Device Property Specification Errors (Table),  
    1959  
Device pushdown, 973  
Device reduction  
    property definition, 1851  
Device reduction effective property language,  
    1851

Device reduction language, see Effective  
    property computation language  
Devices  
    equivalent, 973  
DFM Analyze operation, 220  
    operators, 157  
DFM Assert statement, 228  
DFM CAF operation, 232  
DFM Classify operation, 237  
DFM Connectivity Redundant operation, 243  
DFM Copy operation, 248  
DFM Create Layer operation, 252  
DFM Critical Area operation, 254  
DFM Database statement, 268  
DFM Defaults statement, 272  
DFM Expand Edge operation, 275  
DFM Expand Enclosure operation, 280  
DFM Fill, 285  
DFM Fill operation, 285  
DFM Function statement, 289  
DFM GCA operation, 301  
DFM Grow operation, 304  
DFM Histogram operation, 307  
DFM Measure operation, 311  
DFM NARAC operation, 319  
DFM Operation Errors (table), 1948  
DFM Property Merge operation, 333  
DFM Property operation, 321  
DFM RDB operation, 335  
DFM Read operation, 346  
DFM Redundant Vias operation, 349  
DFM Select Check statement, 352  
DFM Shift Edge operation, 354  
DFM Size operation, 357  
DFM Space operation, 362  
DFM Spec Fill Optimizer statement, 390  
DFM Spec Fill Shape statement, 397  
DFM Spec Fill statement, 365  
DFM Spec Spatial Sample statement, 404  
DFM Spec Via Redundancy statement, 406  
DFM Stamp operation, 408  
DFM Summary Report statement, 409  
DFM Text operation, 411  
DFM Transform operation, 413  
DFM Transition

- overview, 415  
DFM Transition operation, 415  
DFM Unselect Check statement, 424  
DFM YS Autostart statement, 425  
Dimensional check operations  
    acute filter, 515, 584, 670  
    angled filter, 520, 588, 674  
    connectivity filter, 514, 583, 669  
    constraint parameters, 510, 578, 665  
    corner filter, 521, 589, 675  
    Enclosure operation, 506  
    External operation, 576  
    Internal operation, 662  
    intersection filter, 523, 591, 677  
    metrics, 511, 579, 666  
    notch keyword, 581  
    obtuse filter, 517, 586, 672  
    orientation filter, 515, 584, 670  
    output parameters, 528, 598, 681  
    parallel filter, 516, 585, 671  
    perpendicular filter, 516, 585, 671  
    polygon containment, 513, 582, 668  
    projection filter, 518, 587, 673  
    reversal parameters, 526, 595  
    space keyword, 581  
Disconnect, 426  
Disconnect operation, 426  
Donut operation, 428  
Drawn Acute operation, 430  
Drawn Angled operation, 431  
Drawn Offgrid operation, 432  
Drawn Skew operation, 433  
DRC, 39  
    DRC Boolean Nosnap45 statement, 434  
    DRC Cell Name statement, 436  
    DRC Cell Text statement, 439  
    DRC Check Map statement, 440  
    DRC Check Map Statement Errors (Table),  
        1940  
    DRC Check Text statement, 453  
    DRC Exclude False Notch statement, 454  
    DRC Incremental Connect statement, 455  
    DRC Incremental Connect Warning statement,  
        459  
    DRC Keep Empty statement, 461  
DRC Magnify Density statement, 462  
DRC Magnify NAR statement, 463  
DRC Magnify Results statement, 464  
DRC Map Text Depth statement, 470  
DRC Map Text Layer statement, 471  
DRC Map Text statement, 468  
DRC Maximum Cell Name Length statement,  
    473  
DRC Maximum Results statement, 474, 476  
DRC Maximum Unattached Label Warnings  
    statement, 475  
DRC Print Area statement, 478  
DRC Print Perimeter statement, 479  
DRC Results Database Libname statement, 488  
DRC Results Database Precision statement,  
    489  
DRC Results Database statement, 480  
DRC Select Check By Layer statement, 492  
DRC Select Check statement, 491  
DRC Summary Report statement, 494  
DRC Tolerance Factor NAR statement, 497  
DRC Tolerance Factor statement, 496  
DRC Unselect Check By Layer statement, 500  
DRC Unselect Check statement, 499
- E —**
- Edge Clusters, 578, 665  
Edge shielding, 1682  
Edge-directed auxiliary operations, 78  
Edge-directed output, 69  
Effective device instance, 1852  
Effective property computation language, 1851  
    consistency checking, 1865  
    corresponding device definitions, 1864  
    effective device instance, 1852  
    input group, 1852  
    vector functions, 1859  
Electrical Rule Check operations  
    Inductance MICheck, 646  
Electrical rule check operations  
    Pathchk, 1293  
Enclose operation, 502  
Enclose Rectangle operation, 505  
Enclosure checks (examples), 2089  
Enclosure operation, 506  
    acute filter, 515

- angled filter, 520  
connectivity filter, 514  
constraint parameters, 510  
corner filter, 521  
intersection filter, 523  
obtuse filter, 517  
orientation filter, 515  
output parameters, 528  
overview, 506  
parallel filter, 516  
perpendicular filter, 516  
polygon containment filter, 513  
projection filter, 518  
reversal parameters, 526  
Environment Variable Errors (Table), 1964  
Environment variables, 1727  
  CALIBRE\_EXIT\_LAYER\_DEFAULT, 441  
Equivalent devices, 973  
ERC  
  Inductance, 646  
ERC Cell Name statement, 530  
ERC Check Text statement, 532  
ERC Errors (Table), 1964  
ERC Keep Empty statement, 533  
ERC Maximum Results statement, 534  
ERC Maximum Vertex statement, 535  
ERC Path Also statement, 536  
ERC Pathchk statement, 537  
ERC Results Database statement, 544  
ERC Select Check statement, 546  
ERC Summary Report statement, 549  
ERC Unselect Check statement, 551  
ERC, defined, 39  
Error layers  
  operations accepting as input, 314  
Error Messages  
  SVRF Error Statement, 2008  
Error messages  
  Calibre file i/o, 2013  
  Calibre GDSII and OASIS input, 2017  
  Calibre PERC TVF errors, 2050  
  Connectivity extraction, 1964  
  Constraint, 1938  
  Continuous nmDRC, 1938  
Density operation, 1943  
Device definition, 1945  
Device Layer operation, 1963  
Device property specification, 1959  
DFM operation errors, 1948  
DRC Check Map statement, 1940  
ERC, 1964  
General file i/o, 1992  
General specification statement, 2003  
Included rule file, 1967  
Layer and variable resolution, 1999  
Layer definition, 1973  
Layer statement, 1989  
Layer type, 2008  
Layout Cell List statement, 1974  
Layout Cell Match Rule statement, 1974  
Litho operation, 1981  
Location, 1975  
LVS Annotate Devices, 1973  
LVS Cell List statement, 1938  
LVS connectivity extraction, 2044  
LVS Device Type statement, 1963  
LVS Filter statement, 1966  
LVS Map Device statement, 1983  
LVS netlist compilation, 2049  
LVS PERC Load statement, 1977  
LVS PERC Property Statement Errors, 1979, 1981  
LVS Property Initialize statement, 1975  
LVS Property Map statement, 1977  
LVS Recognize Gates Tolerance statement, 2001  
LVS Reduce statement, 1979  
LVS Spice Rename Parameter statement, 2005  
LVS Split Gate Ratio statement, 2001  
LVS Stamp, 2048, 2049  
Macro, 1983  
Net Area Ratio, 1985  
Numeric, 1987  
OPC operation, 1989  
Operation input, 1967  
Parasitic extraction, 1992  
PEX property computation, 1997  
PEX statement, 1996

Pre-processor directive, 1943  
Rule check group, 1967  
Rule check statement, 1992  
Rule file, 2062  
Rule file compilation, 1937  
Runtime, 2010  
Secondary keyword, 1971  
Syntax, 2005  
Trace Property statement, 2006  
Variable resolution, 1999  
Variable statement, 2008  
Error severity handling, 729  
Error-directed auxiliary operations, 79  
Error-directed output, 69  
Exceptions, 729  
Exclude Acute statement, 552  
Exclude Angled statement, 553  
Exclude Cell statement, 554  
Exclude Offgrid statement, 555  
Exclude Skew statement, 556  
Expand Cell statement, 557  
Expand Cell Text statement, 558  
Expand Edge operation, 560  
Expand Text operation, 564  
Expression evaluation failure (DFM Property), 1841  
Extension checks (example), 2089  
Extent Cell operation, 567  
Extent Drawn operation, 571  
Extent generation operations  
    Extent, 565  
    Extent Cell, 567  
    Extent Drawn, 571  
    Extents, 574  
Extent operation, 565  
Extents operation, 574  
Exterior cycle, 428  
External  
    polygon containment filter, 582  
External operation, 576  
    acute filter, 584  
    angled filter, 588  
    connectivity filter, 583  
    constraint parameters, 578  
    corner filter, 589

intersection filter, 591  
notch and space keywords, 581  
obtuse filter, 586  
orientation filter, 584  
output parameters, 598  
overview, 577  
parallel filter, 585  
perpendicular filter, 585  
projection filter, 587  
reversal parameters, 595

— F —

Files  
    instance position, 1500  
Fill  
    defining, 365  
Flag Acute statement, 601  
Flag Angled statement, 603  
Flag Nonsimple Path statement, 607  
Flag Nonsimple statement, 605  
Flag Offgrid statement, 609  
Flag Skew statement, 611  
Flat versus hierarchical processing  
    DFM Grow, 305  
Flatten Cell statement, 614  
Flatten Inside Cell statement, 615  
Flatten operation, 613  
Fracture HITACHI operation, 616  
Fracture JEOL operation, 618  
Fracture MEBES operation, 620  
Fracture NUFLARE operation, 622, 624  
Fracture VBOASIS operation, 627, 629  
Fracturing statements, 82

— G —

GDSII/OASIS cell and placement exceptions (Table), 2015  
General File I/O Errors (Table), 1992  
General Specification Statement Errors (Table), 2003  
Generic LVS Reduce statement, 998  
Geometrically equal, 709  
Group statement, 631  
Grow operation, 632

— H —

Hcell statement, 637  
Hierarchical instance position files, 1500  
Holes operation, 640  
HSIM position files, 1500

— I —

ICrules input warnings (Table), 2042  
ICrules notes (Table), 2042  
ICverify applications, 2115  
Include statement, 644  
Included Rule File Errors (Table), 1967  
Inductance MICheck operation, 646  
Inductance Wire, 649  
Inductance Wire specification, 649  
Input group for effective property computation, 1852  
Input pair for trace property computation, 1873  
Inside also parameter, 526, 595  
Inside Cell operation, 652  
Inside Edge operation, 656  
Inside operation, 650  
Interact operation, 658  
Interior cycle, 428  
Internal  
  polygon containment filter, 668

Internal operation, 662  
  acute filter, 670  
  angled filter, 674  
  connectivity filter, 669  
  constraint parameters, 665  
  corner filter, 675  
  intersection filter, 677  
  obtuse filter, 672  
  orientation filter, 670  
  output parameter, 681  
  overview, 662  
  parallel filter, 671  
  perpendicular filter, 671  
  projection filter, 673

Interval constraints, 45

— K —

Keyword abbreviations, 48

— L —

Label Order operation, 683  
Layer and Variable Resolution Errors (Table), 1999  
Layer constructors described, 69  
Layer Definition Errors (Table), 1973  
Layer Directory statement, 689  
Layer Map statement, 691  
Layer of origin, 70  
Layer operations, 55, 69  
  auxiliary operations  
    edge-directed, 78  
    error-directed, 79  
    miscellaneous, 80  
    polygon-directed, 72  
  edge-directed output, 69  
  error-directed output, 69  
  layer constructors described, 69  
  layer of origin, 70  
  layer selectors described, 69  
  polygon-directed output, 69  
  *see also* Operations  
Layer Resolution statement, 696  
Layer selectors described, 69  
Layer sets, 53  
Layer statement, 685  
  in place of Boolean operation, 688  
Layer Statement Errors (Table), 1989  
Layer Type Errors (Table), 2008  
Layout Allow Duplicate Cell statement, 697  
Layout Base Cell statement, 698  
Layout Base Layer statement, 699  
Layout Bump2 statement, 701  
Layout Case statement, 703  
Layout Cell List Statement Errors (Table), 1974  
Layout Cell Match Rule statement, 709  
Layout Cell Match Rule Statement Errors (Table), 1974  
Layout Clone By Layer statement, 712  
Layout Clone Rotated Placement statement, 714  
Layout Clone Transformed Placement statement, 716  
Layout Depth statement, 719

- Layout Error On Input statement, [720](#)  
Layout grid step size, [1633](#)  
Layout Input Exception RDB statement, [725](#)  
Layout Input Exception Severity statement, [729](#)  
Layout Magnify statement, [750](#)  
Layout Merge On Input statement, [753](#)  
Layout MOS Swappable Properties Errors (Table), [1974](#)  
Layout Path statement, [755](#)  
Layout Path2 statement, [761](#)  
Layout Polygon statement, [765, 767](#)  
Layout Precision statement, [769](#)  
Layout Preserve Case statement, [771](#)  
Layout Preserve Cell List statement, [774](#)  
Layout Primary statement, [775](#)  
Layout Primary2 statement, [777](#)  
Layout Process Box Record statement, [779](#)  
Layout Process Node Record statement, [780](#)  
Layout Property Audit statement, [781](#)  
Layout Property Text OASIS statement, [784](#)  
Layout Property Text statement, [783](#)  
Layout Rename Cell statement, [785](#)  
Layout Rename ICV statement, [786](#)  
Layout Rename Text statement, [787](#)  
Layout System statement, [796](#)  
Layout System2 statement, [802](#)  
Layout Text File statement, [806](#)  
Layout Text statement, [804](#)  
Layout Top Layer statement, [808](#)  
Layout Use Database Precision statement, [810](#)  
Layout Windel Cell statement, [813](#)  
Layout Windel Layer statement, [815](#)  
Layout Windel statement, [811](#)  
Layout Window Cell statement, [818](#)  
Layout Window Clip statement, [820](#)  
Layout Window Layer statement, [821](#)  
Layout Window statement, [816](#)  
Length operation, [823](#)  
Litho File statement, [826](#)  
Litho operation  
    DenseOPC, [824](#)  
    OPC, [827](#)  
    OPCverify, [829](#)  
    ORC, [831](#)  
PSMgate, [835](#)  
Litho Operation Errors (Table), [1981](#)  
Litho Printimage operation, [833](#)  
Location Errors (Table), [1975](#)  
LVS, [39](#)  
LVS Abort On ERC Error statement, [836](#)  
LVS Abort On Softchk statement, [837](#)  
LVS Abort On Supply Error statement, [838](#)  
LVS All Capacitor Pins Swappable statement, [840](#)  
LVS Annotate Devices, [841](#)  
LVS Annotate Devices Errors (Table), [1973](#)  
LVS Auto Expand Hcells statement, [843](#)  
LVS Box statement, [846](#)  
LVS Built-in Device Pin Swap statement, [850](#)  
LVS Built-in MOS NRD\_NRS statement, [852](#)  
LVS Cell List statement, [854](#)  
LVS Cell List statement errors (Table), [1938](#)  
LVS Cell Supply statement, [858](#)  
LVS Center Device Pins, [860](#)  
LVS Check Port Names statement, [863](#)  
LVS Compare Case statement, [865](#)  
LVS Comparison Messages (Table), [2048](#)  
LVS Component Subtype Property statement, [868](#)  
LVS Component Type Property statement, [869](#)  
LVS connectivity extraction errors and warnings (Table), [2044](#)  
LVS Cpoint statement, [870](#)  
LVS Device Type statement, [874](#)  
LVS Device Type Statement Errors (Table), [1963](#)  
LVS Discard Pins By Device statement, [880](#)  
LVS Downcase Device statement, [883](#)  
LVS EDDM Process M statement, [885](#)  
LVS Exact Subtypes statement, [888](#)  
LVS Exclude Hcell statement, [890](#)  
LVS Execute ERC statement, [891](#)  
LVS Expand Seed Promotions statement, [892](#)  
LVS Expand Unbalanced Cells statement, [893](#)  
LVS Filter statement, [895](#)  
LVS Filter Statement Errors (Table), [1966](#)  
LVS Filter Unused Bipolar statement, [900](#)  
LVS Filter Unused Capacitors statement, [901](#)  
LVS Filter Unused Diodes statement, [902](#)

- LVS Filter Unused MOS statement, [903](#)  
LVS Filter Unused Option statement, [904](#)  
LVS Filter Unused Resistors statement, [910](#)  
LVS Flatten Inside Cell statement, [911](#)  
LVS Global Layout Name statement, [914](#)  
LVS Globals Are Ports statement, [917](#)  
LVS Ground Name statement, [918](#)  
LVS Heap Directory statement, [920](#)  
LVS Ignore Ports statement, [922](#)  
LVS Ignore Trivial Named Ports statement, [923](#)  
LVS Inject Logic statement, [925](#)  
LVS Isolate Shorts statement, [927](#)  
LVS Map Device statement, [938](#)  
LVS Map Device Statement Errors (Table), [1983](#)  
LVS MOS Swappable Properties Statement, [941](#)  
LVS Netlist All Texted Pins statement, [942](#)  
LVS Netlist Allow Inconsistent Model statement, [943](#)  
LVS Netlist Box Contents statement, [945](#)  
LVS Netlist Comment Coded Properties statement, [946](#)  
LVS Netlist Comment Coded Substrate statement, [948](#)  
LVS Netlist Unnamed Box Pins statement, [949](#)  
LVS NL Pin Locations statement, [950](#)  
LVS Non User Name statement, [951](#)  
LVS Out Of Range Exclude Zero statement, [954](#)  
LVS PERC Load Statement Errors (Table), [1977](#)  
LVS PERC Property Statement Errors (Table), [1979, 1981](#)  
LVS Pin Name Property statement, [956](#)  
LVS Power Name statement, [957](#)  
LVS Precise Interaction statement, [959](#)  
LVS Preserve Box Cells statement, [960](#)  
LVS Preserve Box Ports statement, [962](#)  
LVS Preserve Floating Top Nets statement, [964](#)  
LVS Preserve Parameterized Cells statement, [965](#)  
LVS Property Initialize built-in language, [1867](#)  
functions, [1871](#)  
LVS Property Initialize statement, [966](#)  
LVS Property Initialize Statement Errors (Table), [1975](#)  
LVS Property Map statement, [968](#)  
LVS Property Map Statement Errors (Table), [1977](#)  
LVS Property Resolution Maximum statement, [971](#)  
LVS Push Devices statement, [972](#)  
LVS Recognize Gates statement, [986](#)  
LVS Recognize Gates Tolerance statement, [990](#)  
LVS Recognize Gates Tolerance Statement Errors (Table), [2001](#)  
LVS Reduce Parallel Bipolar statement, [1004](#)  
LVS Reduce Parallel Capacitors statement, [1007](#)  
LVS Reduce Parallel Diodes statement, [1010](#)  
LVS Reduce Parallel MOS statement, [1013](#)  
LVS Reduce Parallel Resistors statement, [1016](#)  
LVS Reduce Semi Series MOS statement, [1019](#)  
LVS Reduce Series Capacitors statement, [1020](#)  
LVS Reduce Series MOS statement, [1023](#)  
LVS Reduce Series Resistors statement, [1026](#)  
LVS Reduce Split Gates statement, [1029](#)  
LVS Reduce statement, [995](#)  
LVS Reduce Statement Errors (Table), [1979](#)  
LVS Reduction Priority, [1033](#)  
LVS Report Maximum statement, [1036](#)  
LVS Report Option statement, [1037](#)  
LVS Report statement, [1035, 1106](#)  
LVS Report Units statement, [1048](#)  
LVS Report Warnings Hcell Only statement, [1050](#)  
LVS Report Warnings Top Only statement, [1051](#)  
LVS Reverse WL statement, [1052](#)  
LVS Short Equivalent Nodes statement, [1053](#)  
LVS Show Seed Promotions Maximum statement, [1059](#)  
LVS Show Seed Promotions statement, [1055](#)  
LVS Signature Maximum statement, [1060](#)  
LVS Soft Substrate Pins statement, [1061](#)

- LVS Softchk statement, [1062](#)  
LVS Spice Allow Duplicate Subcircuit Names statement, [1065](#)  
LVS Spice Allow Floating Pins statement, [1066](#)  
LVS Spice Allow Inline Parameters statement, [1067](#)  
LVS Spice Allow Unquoted Strings statement, [1070](#)  
LVS Spice Conditional LDD, [1071](#)  
LVS Spice Cull Primitive Subcircuits, [1072](#)  
LVS Spice Implied MOS Area statement, [1074](#)  
LVS Spice Multiplier Name statement, [1075](#)  
LVS Spice Option statement, [1077](#)  
LVS Spice Override Globals statement, [1079](#)  
LVS Spice Prefer Pins statement, [1082](#)  
LVS Spice Redefine Param statement, [1083](#)  
LVS Spice Rename Parameter Statement, [1085](#)  
LVS Spice Rename Parameter Statement Errors (Table), [2005](#)  
LVS Spice Replicate Devices statement, [1091](#)  
LVS Spice Scale X Parameters statement, [1094](#)  
LVS Spice Slash Is Space statement, [1096](#)  
LVS Spice Strict WL statement, [1097](#)  
LVS Split Gate Ratio statement, [1099](#)  
LVS Split Gate Ratio Statement Errors (Table), [2001](#)  
LVS Stamp errors (Table), [2049](#)  
LVS Strict Subtypes statement, [1105](#)  
LVS Write Injected Layout Netlist statement, [1107](#)  
LVS Write Injected Source Netlist statement, [1109](#)  
LVS Write Layout Netlist statement, [1111](#)  
LVS Write Source Netlist statement, [1113](#)  
LVS-actionable device properties, [978](#)
- M —
- Macro Errors (Table), [1983](#)  
Macros, [64](#)  
Magnify operation, [1115](#)  
Mask data prep operations  
Fracture HITACHI, [616](#)  
Fracture JEOL, [618](#)  
Fracture MEBES, [620](#)  
Fracture NUFLARE, [622](#), [624](#)
- Fracture VBOASIS, [627](#), [629](#)  
MDP EMBED, [1130](#)  
MDP Mapsize, [1132](#)  
MDPmerge, [1143](#)  
MDPstat, [1144](#)  
MDPverify, [1145](#)  
Mask Results Database statement, [1116](#)  
Mask SVDB Directory statement, [1118](#)  
Maximum results and polygon segmentation warnings (Table), [2056](#)  
MDP, [39](#)  
MDP CHECKMAP syntax, [1128](#)  
MDP CHECKMAP operation, [1128](#)  
MDP EMBED operation, [1130](#)  
MDP Mapsize syntax, [1132](#)  
MDP Mapsize operation, [1132](#)  
MDP Maskopt syntax, [1137](#)  
MDP Maskopt operation, [1137](#), [1140](#)  
MDP Oasis\_Extent, [1140](#)  
MDPmerge operation, [1143](#)  
MDPstat operation, [1144](#)  
MDPverify operation, [1145](#)  
Measure all parameter, [513](#), [582](#), [668](#)  
Measure coincident parameter, [513](#), [668](#)  
Measurement regions, [511](#), [579](#), [666](#)  
Merge operation, [1147](#)  
Methodology operations  
Pins, [1591](#)  
Port Layer Polygon, [1596](#)  
Port Layer Text, [1599](#)  
Ports, [1601](#)  
Topex, [1692](#)  
Metrics, [511](#), [579](#), [666](#)  
Miscellaneous warnings (Table), [2056](#)
- N —
- Names, [43](#)  
NARAC defined, [1161](#)  
NARAC layer, [1156](#)  
Negative numbers, specifying as coordinates, [46](#)  
Net Area operation, [1151](#)

Net Area Ratio Accumulate operation, 1173  
Net Area Ratio Errors (Table), 1985  
Net Area Ratio operation, 1152  
Net Area Ratio Print operation, 1177  
Net Interact, 1178  
Net operation, 1149  
Node-preserving operations, *also* net-preserving operations, 70  
Non-orientable polygon, 605  
Not acute parameter, 515, 584, 670  
Not Angle operation, 1181  
Not Area operation, 1183  
Not Coincident Edge operation, 1184  
Not Coincident Inside Edge operation, 1185  
Not Coincident Outside Edge operation, 1186  
Not connected parameter, 514, 583, 669  
Not corner parameter, 521, 589, 675  
Not Cut operation, 1187  
Not Donut operation, 1189  
Not Enclose operation, 1190  
Not Enclose Rectangle operation, 1192  
Not Inside Cell operation, 1195  
Not Inside Edge operation, 1198  
Not Inside operation, 1193  
Not Interact operation, 1199  
Not Length operation, 1202  
Not Net operation, 1203  
Not obtuse parameter, 517, 586, 672  
NOT operation, 1180  
Not Outside Edge operation, 1205  
Not Outside operation, 1204  
Not parallel parameter, 516, 585, 671  
Not perpendicular parameter, 516, 585, 671  
Not projecting parameter, 518, 587, 673  
Not Rectangle operation, 1206  
Not Touch Edge operation, 1211  
Not Touch Inside Edge operation, 1213  
Not Touch operation, 1209  
Not Touch Outside Edge operation, 1215  
Not With Edge operation, 1217  
Not With Neighbor operation, 1218  
Not With Text operation, 1223  
Not With Width operation, 1224  
Notch keyword, 581  
Note messages

ICrules, 2042  
Numeric Errors (Table), 1987  
Numeric expressions, 46  
**— O —**  
Obtuse also parameter, 517, 586, 672  
Obtuse only parameter, 517, 586, 672  
Offgrid operation, 1225  
OPC Operation Errors (Table), 1989  
OPC operations  
    Opcbias, 1235  
    Opclineend, 1245  
    Opcsbar, 1255  
    RET NMDPC, 1634, 1639  
    RET PXOPC, 1636  
    RET SRAF\_Fill, 1641  
    Opcbias operation, 1235  
    Opclineend operation, 1245  
    Opcsbar operation, 1255  
Operation Input Errors (Table), 1967  
Operations  
    AND, 115  
    Angle, 118  
    Area, 120  
    Attach, 121  
    Coincident Edge, 126  
    Coincident Inside Edge, 127  
    Coincident Outside Edge, 128  
    Connect, 129  
    Convex Edge, 134  
    Copy, 138  
    Cut, 141  
    DBCLASSIFY, 144  
    Deangle, 144, 154  
    defined, 40  
    Density, 156  
    Density Convolve, 183  
    Device, 196  
    Device Layer, 216  
    DFM Analyze, 220  
    DFM CAF, 232  
    DFM Classify, 237  
    DFM Connectivity Redundant, 243  
    DFM Copy, 248  
    DFM Create Layer, 252  
    DFM Critical Area, 254

- DFM Expand Edge, 275  
DFM Expand Enclosure, 280  
DFM Fill, 285  
DFM GCA, 301  
DFM Grow, 304  
DFM Histogram, 307  
DFM Measure, 311  
DFM NARAC, 319  
DFM Property, 321  
DFM Property Merge, 333  
DFM RDB, 335  
DFM Read, 346  
DFM Redundant Vias, 349  
DFM Shift Edge, 354  
DFM Size, 357  
DFM Space, 362  
DFM Stamp, 408  
DFM Text, 411  
DFM Transform, 413  
DFM Transition, 415  
Disconnect, 426  
Donut, 428  
Drawn Acute, 430  
Drawn Angled, 431  
Drawn Offgrid, 432  
Drawn Skew, 433  
Enclose, 502  
Enclose Rectangle, 505  
Enclosure, 506  
Expand Edge, 560  
Expand Text, 564  
Extent, 565  
Extent Cell, 567  
Extent Drawn, 571  
Extents, 574  
External, 576  
Flatten, 613  
Grow, 632  
Holes, 640  
Inductance MICheck, 646  
Inside, 650  
Inside Cell, 652  
Inside Edge, 656  
Interact, 658  
Internal, 662  
Length, 823  
Litho nmOPC, 824  
Litho OPC, 827  
Litho OPCVerify, 829  
Litho ORC, 831  
Litho Printimage, 833  
Litho PSMgate, 835  
Magnify, 1115  
Merge, 1147  
Net, 1149  
Net Area, 1151  
Net Area Ratio, 1152  
Net Area Ratio Accumulate, 1173  
Net Area Ratio Print, 1177  
Net Interact, 1178  
NOT, 1180  
Not Angle, 1181  
Not Area, 1183  
Not Coincident Edge, 1184  
Not Coincident Inside Edge, 1185  
Not Coincident Outside Edge, 1186  
Not Cut, 1187  
Not Donut, 1189  
Not Enclose, 1190  
Not Enclose Rectangle, 1192  
Not Inside, 1193  
Not Inside Cell, 1195  
Not Inside Edge, 1198  
Not Interact, 1199  
Not Length, 1202  
Not Net, 1203  
Not Outside, 1204  
Not Outside Edge, 1205  
Not Rectangle, 1206  
Not Touch, 1209  
Not Touch Edge, 1211  
Not Touch Inside Edge, 1213  
Not Touch Outside Edge, 1215  
Not With Edge, 1217  
Not With Neighbor, 1218  
Not With Text, 1223  
Not With Width, 1224  
Offgrid, 1225  
Opcbias, 1235  
Opclineend, 1245

- Opchsbar, 1255  
OR, 1281  
OR Edge, 1283  
Ornet, 1285  
Outside, 1286  
Outside Edge, 1287  
Path Length, 1291  
Pathchk, 1293  
Perimeter, 1319  
Pins, 1591  
Ports, 1601  
Push, 1605  
Rectangle, 1608  
Rectangle Enclosure, 1611  
Rectangles, 1618  
RET NMDPC, 1634, 1639  
RET PXOPC, 1636  
RET SRAF\_Fill, 1641  
Rotate, 1643  
Shift, 1651  
Shrink, 1653  
Size, 1656  
Snap, 1664  
TDDRC, 1678  
Topex, 1692  
Touch, 1693  
Touch Edge, 1696  
Touch Inside Edge, 1698  
Touch Outside Edge, 1700  
TVF, 1908  
Vertex, 1732  
With Edge, 1755  
With Neighbor, 1757  
With Text, 1762  
With Width, 1763  
XOR, 1768  
Opposite extended Fsymmetric metric, 512, 580, 667  
Opposite extended metric, 511, 579, 666  
Opposite extended symmetric metric, 512, 580, 667  
Opposite extended1 metric, 512, 580, 667  
Opposite extended2 metric, 512, 580, 667  
Opposite fsymmetric metric, 512, 580, 667  
Opposite metric, 511, 579, 666  
Opposite symmetric metric, 512, 580, 667  
Opposite1 metric, 512, 580, 667  
Opposite2 metric, 512, 580, 667  
OR Edge operation, 1283  
OR operation, 1281  
Ornet operation, 1285  
Other nmDRC checks (examples), 2109  
Output  
    DFM Grow, 305  
    DFM Transition, 415, 417  
Output file open warnings (Table), 2058  
OUTPUT1 keyword, 422  
OUTPUT2 keyword, 422  
OUTPUT3 keyword, 422  
Outside also parameter, 526  
Outside Edge operation, 1287  
Outside operation, 1286  
Overlap parameter, 523, 592, 678
- P —**
- Parallel also parameter, 516, 585, 671  
Parallel only parameter, 516, 585, 671  
Parasitic Extraction Errors (Table), 1992  
Parasitic extraction specifications  
    Capacitance Order, 124  
    Parasitic Variation, 1288  
    Resistance Connection, 1625  
    Resistance Device\_Seed, 1628  
    Resistance Rho, 1629  
    Resistance Sheet, 1631  
Parasitic Variation statement, 1288  
Path Length operation, 1291  
Pathchk operation, 1293  
PERC  
    Initialization procedures, 1304  
    Netlist transformation, 1304  
    Processing Order of PERC Load statements, 1305  
    Rule check procedures, 1304  
PERC Load statement, 1304  
PERC Netlist statement, 1308  
PERC Pattern Path statement, 1310  
PERC Property statement, 1312  
PERC Report Maximum statement, 1315  
PERC Report Option statement, 1316

PERC Report Placement List Maximum statement, [1317](#)  
PERC Report statement, [1314](#)  
PERC Waiver Path statement, [1318](#)  
Perimeter operation, [1319](#)  
Perpendicular also parameter, [516, 585, 671](#)  
Perpendicular only parameter, [516, 585, 671](#)  
PEX Alias statement, [1322](#)  
PEX BA Mapfile statement, [1327](#)  
PEX Bulk Layer statement, [1320, 1335](#)  
PEX CMP Mode statement, [1339](#)  
PEX Contact Capacitance statement, [1341](#)  
PEX Corner Custom statement, [1343](#)  
PEX Corner statement, [1342](#)  
PEX DEF Extract Cell Obstructions statement, [1346](#)  
PEX DEF Map statement, [1347, 1456](#)  
PEX Density Estimate statement, [1348](#)  
PEX Density Window statement, [1349](#)  
PEX Driver File statement, [1351](#)  
PEX Elayer statement, [1352](#)  
PEX Exclude Distributed statement, [1353](#)  
PEX Exclude Lumped statement, [1355](#)  
PEX Extract Exclude statement, [1357](#)  
PEX Extract Floating Nets statement, [1359](#)  
PEX Extract Include statement, [1362](#)  
PEX Extract Rgate statement, [1364](#)  
PEX Extract Temperature statement, [1369](#)  
PEX Fieldsolver Boundary statement, [1371](#)  
PEX Fieldsolver Cell\_array statement, [1376](#)  
PEX Fieldsolver Endcap Spacing statement, [1385](#)  
PEX Fieldsolver Mode statement, [1386](#)  
PEX Fill Handling statement, [1389](#)  
PEX Fill Model statement, [1390](#)  
PEX Fracture Frequency statement, [1391](#)  
PEX Generate Driver File Tag statement, [1392](#)  
PEX Generate Driver\_File Tag statement, [1393](#)  
PEX Ground Layer statement, [1395](#)  
PEX Ground statement, [1394](#)  
PEX Ideal Xcell statement, [1396](#)  
PEX Ignore Capacitance statement, [1398](#)  
PEX Ignore Resistance statement, [1410, 1411](#)  
PEX Include Distributed statement, [1413](#)  
PEX Include Lumped statement, [1415](#)

PEX Indie Spacing statement, [1417](#)  
PEX Inductance ... Frequency statement, [1431](#)  
PEX Inductance ... Switch Time statement, [1445](#)  
PEX Inductance ... Switch\_Time statement, [1447](#)  
PEX Inductance Default Partial Model statement, [1418](#)  
PEX Inductance Default PI statement, [1419](#)  
PEX Inductance Differential Pair statement, [1420](#)  
PEX Inductance Doprocess statement, [1422](#)  
PEX Inductance Driver File statement, [1423](#)  
PEX Inductance Driver Summary statement, [1424](#)  
PEX Inductance Extract Layers statement, [1425](#)  
PEX Inductance Filter statement, [1426](#)  
PEX Inductance Forward Coupling statement, [1429](#)  
PEX Inductance Frequency statement, [1430](#)  
PEX Inductance Layer Summary statement, [1433](#)  
PEX Inductance MICHECK Constraint statement, [1434](#)  
PEX Inductance Minlength statement, [1435](#)  
PEX Inductance Parameters statement, [1436](#)  
PEX Inductance Range statement, [1437](#)  
PEX Inductance Returnpath statement, [1438](#)  
PEX Inductance Same Net Mutual statement, [1440](#)  
PEX Inductance Self statement, [1441](#)  
PEX Inductance Skin Include statement, [1442](#)  
PEX Inductance Switch Time statement, [1443](#)  
PEX Inductance Switch\_Time statement, [1444](#)  
PEX Inductance Victim File statement, [1453](#)  
PEX Inductance Victim statement, [1448](#)  
PEX Inductance Victim\_File statement, [1454](#)  
PEX Inductance Victim\_Path statement, [1450, 1452](#)  
PEX Magnify statement, [1455](#)  
PEX Netlist ADMS statement, [1472](#)  
PEX Netlist Capacitance Unit statement, [1475](#)  
PEX Netlist Character Map statement, [1476](#)

- PEX Netlist Connection Section statement, [1478](#)  
PEX Netlist Create Smashed Device Names statement, [1479](#)  
PEX Netlist Device Resistance Model statement, [1480](#)  
PEX Netlist Distributed statement, [1481](#)  
PEX Netlist Escape Characters statement, [1487](#)  
PEX Netlist Export Ports statement, [1488](#)  
PEX Netlist Filter statement, [1489](#)  
PEX Netlist Global Nets statement, [1491](#)  
PEX Netlist Linewrap statement, [1492](#)  
PEX Netlist LPE Ignore Idealnet statement, [1493](#)  
PEX Netlist LPE Using Extmode statement, [1494](#)  
PEX Netlist Lumped statement, [1495](#)  
PEX Netlist Mutual Resistance statement, [1498](#)  
PEX Netlist Noxref Net Names statement, [1499](#)  
PEX Netlist Position File statement, [1500](#)  
PEX Netlist Replicated\_Device Delimiter statement, [1501](#)  
PEX Netlist SchematicOnly statement, [1503](#)  
PEX Netlist Select File statement, [1508](#)  
PEX Netlist Simple statement, [1515](#)  
PEX Netlist Smashed\_Device Delimiter statement, [1517](#)  
PEX Netlist statement, [1458](#)  
PEX Netlist Subnode Section statement, [1518](#)  
PEX Netlist Unshort Device Pins statement, [1519](#)  
PEX Netlist Uppercase Keywords statement, [1520](#)  
PEX Netlist Virtual Connect statement, [1521](#)  
PEX Pin Order statement, [1522](#)  
PEX Power statement, [1524](#)  
PEX Probe File statement, [1525](#)  
PEX Property Computation Errors (Table), [1997](#)  
PEX Reduce Analog statement, [1527](#)  
PEX Reduce CC statement, [1529](#)  
PEX Reduce Digital statement, [1531](#)  
PEX Reduce Distributed statement, [1533](#)  
PEX Reduce Mincap statement, [1534](#)  
PEX Reduce Minres statement, [1536](#)  
PEX Reduce ROnly statement, [1538](#)  
PEX Reduce TICER statement, [1540](#)  
PEX Reduce Via Resistance statement, [1544](#)  
PEX Report Coupling Capacitance statement, [1549](#)  
PEX Report Distributed statement, [1551](#)  
PEX Report Lumped statement, [1553](#)  
PEX Report Mutual Inductance statement, [1554](#)  
PEX Report Netsummary statement, [1556](#)  
PEX Report Point2Point statement, [1559](#)  
PEX Report statement, [1548](#)  
PEX Resistance Parameters statement, [1563](#)  
PEX Sensitivity statement, [1570](#)  
PEX Skin Include statement, [1571](#)  
PEX Slots Handling statement, [1572](#)  
PEX Statement Errors (Table), [1996](#)  
PEX Temperature statement, [1574](#)  
PEX Thickness EQN statement, [1575](#)  
PEX Thickness Nominal statement, [1577](#)  
PEX Threshold statement, [1578](#)  
PEX Tolerance Distributed statement, [1580](#)  
PEX Via Capacitance statement, [1582](#)  
PEX Via Reduction Resistance statement, [1583](#)  
PEX Xcell Extract Mode statement, [1586](#)  
PEX Xcell Precedence statement, [1588](#)  
PEX Xcell statement, [1584](#)  
PEX, defined, [39](#)  
Pins operation, [1591](#)  
Placement of vias, [422](#)  
Polygon flagging warnings (Table), [2060](#)  
Polygon statement, [1592](#)  
Polygon-directed auxiliary operations, [72](#)  
Polygon-directed output, [69](#)  
Port Depth statement, [1594](#)  
Port Layer Polygon statement, [1596](#)  
Port Layer Text statement, [1599](#)  
Ports operation, [1601](#)  
Precision statement, [1602](#)  
    database units, [1602](#)  
Pre-Processor Directive Errors (Table), [1943](#)  
Pre-processor directives, [56](#)  
PrimeTime, [1459](#), [1473](#), [1482](#)  
Primitive layer number, [701](#)

Process variables  
  \$precision, 1603  
  \$unit\_length, 1722  
Projecting parameter, 518, 587, 673  
Push Cell statement, 1607  
Push operation, 1605

— R —

Rectangle Enclosure operation, 1611  
Rectangle operation, 1608  
Rectangles operation, 1618  
Redundant transition polygons, 417  
Region centerline parameter, 528, 598, 681  
Region extents parameter, 528, 598, 681  
Region parameter, 528, 598, 681  
Resistance Connection statement, 1625  
Resistance Device\_Seed statement, 1628  
Resistance Rho statement, 1629  
Resistance Sheet statement, 1631  
Resolution statement, 1633  
  layout grid step size, 1633  
Results database, 335  
RET, 39  
RET NMDPC operation, 1634, 1639  
RET PXOPC operation, 1636  
RET SRAF\_Fill operation, 1641  
Rotate operation, 1643  
Rule Check Group Errors (Table), 1967  
Rule Check Statement Errors (Table), 1992  
Rule file errors (Table), 2062  
Rule files, 2115  
  abbreviations, keyword, 48  
  compilation overview, 41  
  constraints, 45  
  described, 40  
  elements summary, 69  
  error and warning messages, 1937  
  example, 2069  
  including another rule file, 644  
  keyword abbreviations, 48  
  macros, 64  
  numeric expressions, 46  
  string constants, 46  
  structure, 52  
  variables in Pyxis Layout, 47  
Runtime error and warning messages, 2010

Runtime TVF, 1906

— S —

SOFFSET  
  and negative scattering bars, 1271  
  and positive scattering bars, 1271  
Sconnect operation, 1644  
Secondary Keyword Errors (Table), 1971  
Seed layer, 200  
Seed promotion, 973  
Severities, 729  
Shallow trench isolation device calculations, 1833  
Shielding  
  edges, 1682  
Shift operation, 1651  
Shorts  
  single layer, 259, 262  
Shrink operation, 1653  
Simple layer, 691  
Singular parameter  
  Enclosure operation, 524  
  External operation, 592  
  Internal operation, 678  
  Rectangle Enclosure, 1611  
Size operation, 1656  
Snap Offgrid statement, 1665  
Snap operation, 1664  
Soft boundary, 1134  
Source Case statement, 1666  
Source Path statement, 1668  
Source Primary statement, 1670  
Source System statement, 1671  
SPACE condition, 316  
Space keyword, 581  
Spacing check examples, 2076  
Sparse OPC, 827  
Specification statements  
  defined, 40, 55  
  DFM Assert, 228  
  DFM Database, 268  
  DFM Defaults, 272  
  DFM Function, 289  
  DFM Select Check, 352  
  DFM Spec Fill, 365  
  DFM Spec Fill Optimizer, 390

DFM Spec Fill Shape, 397  
DFM Spec Spatial Sample, 404  
DFM Spec Via Redundancy, 406  
DFM Summary Report, 409  
DFM Unselect Check, 424  
DFM YS Autostart, 425  
DRC Boolean Nosnap45, 434  
DRC Cell Name, 436  
DRC Cell Text, 439  
DRC Check Map, 440  
DRC Check Text, 453  
DRC Exclude False Notch, 454  
DRC Incremental Connect, 455  
DRC Incremental Connect Warning, 459  
DRC Keep Empty, 461  
DRC Magnify Density, 462  
DRC Magnify NAR, 463  
DRC Magnify Results, 464  
DRC Map Text, 468  
DRC Map Text Depth, 470  
DRC Map Text Layer, 471  
DRC Maximum Cell Name Length, 473  
DRC Maximum Results, 474  
DRC Maximum Unattached Label  
    Warnings, 475  
DRC Maximum Vertex, 476  
DRC Print Area, 478  
DRC Print Perimeter, 479  
DRC Results Database, 480  
DRC Results Database Libname, 488  
DRC Results Database Precision, 489  
DRC Select Check, 491  
DRC Select Check By Layer, 492  
DRC Summary Report, 494  
DRC Tolerance Factor, 496  
DRC Tolerance Factor NAR, 497  
DRC Unselect Check, 499  
DRC Unselect Check By Layer, 500  
ERC Cell Name, 530  
ERC Check Text, 532  
ERC Keep Empty, 533  
ERC Maximum Results, 534  
ERC Maximum Vertex, 535  
ERC Path Also, 536  
ERC Pathchk, 537

ERC Results Database, 544  
ERC Select Check, 546  
ERC Summary Report, 549  
ERC Unselect Check, 551  
Exclude Acute, 552  
Exclude Angled, 553  
Exclude Cell, 554  
Exclude Offgrid, 555  
Exclude Skew, 556  
Expand Cell, 557  
Expand Cell Text, 558  
Flag Acute, 601  
Flag Angled, 603  
Flag Nonsimple, 605  
Flag Nonsimple Path, 607  
Flag Offgrid, 609  
Flag Skew, 611  
Flatten Cell, 614  
Flatten Inside Cell, 615  
Group, 631  
Hcell, 637  
Include, 644  
Layer, 685  
Layer Directory, 689  
Layer Map, 691  
Layer Resolution, 696  
Layout Allow Duplicate Cell, 697  
Layout Base Cell, 698  
Layout Base Layer, 699  
Layout Bump2, 701  
Layout Case, 703  
Layout Cell Match Rule, 709  
Layout Clone By Layer, 712  
Layout Clone Rotated Placements, 714  
Layout Clone Transformed Placements,  
    716  
Layout Depth, 719  
Layout Error On Input, 720  
Layout Input Exception RDB, 725  
Layout Input Exception Severity, 729  
Layout Magnify, 750  
Layout Merge On Input, 753  
Layout Path, 755  
Layout Path2, 761  
Layout Polygon, 765, 767

Layout Precision, 769  
Layout Preserve Case, 771  
Layout Preserve Cell List, 774  
Layout Primary, 775  
Layout Primary2, 777  
Layout Process Box Record, 779  
Layout Process Node Record, 780  
Layout Property Audit, 781  
Layout Property Text, 783  
Layout Property Text OASIS, 784  
Layout Rename Cell, 785  
Layout Rename ICV, 786  
Layout Rename Text, 787  
Layout System, 796  
Layout System2, 802  
Layout Text, 804  
Layout Text File, 806  
Layout Top Layer, 808  
Layout Use Database Precision, 810  
Layout Windel, 811  
Layout Windel Cell, 813  
Layout Windel Layer, 815  
Layout Window, 816  
Layout Window Clip, 820  
Layout Window Layer, 821  
Layout WindowCell, 818  
Litho File, 826  
LVS Abort On ERC Error, 836  
LVS Abort On Softchk, 837  
LVS Abort On Supply Error, 838  
LVS All Capacitor Pins Swappable, 840  
LVS Annotate Devices, 841  
LVS Auto Expand Hcells, 843  
LVS Box, 846  
LVS Builtin Device Pin Swap, 850  
LVS Builtin MOS NRD\_NRS, 852  
LVS Cell List, 854  
LVS Center Device Pins, 860  
LVS Check Port Names, 863  
LVS Compare Case, 865  
LVS Component Subtype Property, 868  
LVS Component Type Property, 869  
LVS Cpoint, 870  
LVS Device Type, 874  
LVS Discard Pins By Device, 880

LVS Downcase Device, 883  
LVS EDDM Process M, 885  
LVS Exact Subtypes, 888  
LVS Exclude Hcell, 890  
LVS Execute ERC, 891  
LVS Expand Seed Promotions, 892  
LVS Expand Unbalanced Cells, 893  
LVS Filter, 895  
LVS Filter Unused Bipolar, 900  
LVS Filter Unused Capacitors, 901  
LVS Filter Unused Diodes, 902  
LVS Filter Unused MOS, 903  
LVS Filter Unused Option, 904  
LVS Filter Unused Resistors, 910  
LVS Flatten Inside Cell, 911  
LVS Global Layout Name, 914  
LVS Globals Are Ports, 917  
LVS Ground Name, 918  
LVS Heap Directory, 920  
LVS Ignore Ports, 922  
LVS Inject Logic, 925  
LVS Isolate Shorts, 927  
LVS Map Device, 938  
LVS MOS Swappable Properties, 941  
LVS Netlist All Texted Pins, 942  
LVS Netlist Allow Inconsistent Model, 943  
LVS Netlist Box Contents, 945  
LVS Netlist Comment Coded Properties, 946  
LVS Netlist Comment Coded Substrate, 948  
LVS Netlist Unnamed Box Pins, 949  
LVS NL Pin Locations, 950  
LVS Non User Name, 951  
LVS Out Of Range Exclude Zero, 954  
LVS Pin Name Property, 956  
LVS Power Name, 957  
LVS Precise Interaction, 959  
LVS Preserve Box Cells, 960  
LVS Preserve Box Ports, 962  
LVS Preserve Floating Top Nets, 964  
LVS Preserve Parameterized Cells, 965  
LVS Property Initialize, 966  
LVS Property Map, 968  
LVS Property Resolution Maximum, 971

- LVS Push Devices, [972](#)  
LVS Recognize Gates, [986](#)  
LVS Recognize Gates Tolerance, [990](#)  
LVS Reduce, [995](#)  
LVS Reduce Parallel Bipolar, [1004](#)  
LVS Reduce Parallel Capacitors, [1007](#)  
LVS Reduce Parallel Diodes, [1010](#)  
LVS Reduce Parallel MOS, [1013](#)  
LVS Reduce Parallel Resistors, [1016](#)  
LVS Reduce Semi Series MOS, [1019](#)  
LVS Reduce Series Capacitors, [1020](#)  
LVS Reduce Series MOS, [1023](#)  
LVS Reduce Series Resistors, [1026](#)  
LVS Reduce Split Gates, [1029](#)  
LVS Report, [1035](#), [1106](#)  
LVS Report Maximum, [1036](#)  
LVS Report Option, [1037](#)  
LVS Report Units, [1048](#)  
LVS Report Warnings Hcell Only, [1050](#)  
LVS Report Warnings Top Only, [1051](#)  
LVS Reverse WL, [1052](#)  
LVS Short Equivalent Nodes, [1053](#)  
LVS Show Seed Promotions, [1055](#)  
LVS Show Seed Promotions Maximum, [1059](#)  
LVS Signature Maximum, [1060](#)  
LVS Soft Substrate Pins, [1061](#)  
LVS Softchk, [1062](#)  
LVS Spice Allow Duplicate Subcircuit Names, [1065](#)  
LVS Spice Allow Floating Pins, [1066](#)  
LVS Spice Allow Inline Parameters, [1067](#)  
LVS Spice Allow Unquoted Strings, [1070](#)  
LVS Spice Conditional LDD, [1071](#)  
LVS Spice Cull Primitive Subcircuits, [1072](#)  
LVS Spice Implied MOS Area, [1074](#)  
LVS Spice Multiplier Name, [1075](#)  
LVS Spice Option, [1077](#)  
LVS Spice Override Globals, [1079](#)  
LVS Spice Prefer Pins, [1082](#)  
LVS Spice Redefine Param, [1083](#)  
LVS Spice Rename Parameter, [1085](#)  
LVS Spice Replicate Devices, [1091](#)  
LVS Spice Scale X Parameters, [1094](#)  
LVS Spice Slash Is Space, [1096](#)  
LVS Spice Strict WL, [1097](#)  
LVS Split Gate Ratio, [1099](#)  
LVS Strict Subtypes, [1105](#)  
LVS Write Injected Layout Netlist, [1107](#)  
LVS Write Injected Source Netlist, [1109](#)  
LVS Write Layout Netlist, [1111](#)  
LVS Write Source Netlist, [1113](#)  
Mask Results Database, [1116](#)  
Mask SVDB Directory, [1118](#)  
PERC Load, [1304](#)  
PERC Netlist, [1308](#)  
PERC Pattern Path, [1310](#)  
PERC Property, [1312](#)  
PERC Report, [1314](#)  
PERC Report Maximum, [1315](#)  
PERC Report Option, [1316](#)  
PERC Report Placement List Maximum, [1317](#)  
PERC Waiver Path, [1318](#)  
PEX Netlist Filter, [1489](#)  
Polygon, [1592](#)  
Port Depth, [1594](#)  
Port Layer Polygon, [1596](#)  
Port Layer Text, [1599](#)  
Precision, [1602](#)  
Push Cell, [1607](#)  
Resolution, [1633](#)  
Snap Offgrid, [1665](#)  
Source Case, [1666](#)  
Source Path, [1668](#)  
Source Primary, [1670](#)  
Source System, [1671](#)  
SVRF Error, [1675](#)  
SVRF Message, [1676](#)  
SVRF Version, [1677](#)  
Text, [1685](#)  
Text Depth, [1687](#)  
Text Layer, [1689](#)  
Text Print Maximum, [1690](#)  
Title, [1691](#)  
Trace Property, [1702](#)  
Unit Capacitance, [1718](#)  
Unit Inductance, [1720](#)  
Unit Length, [1722](#)  
Unit Resistance, [1724](#)

Unit Time, [1726](#)  
Variable, [1727](#)  
Virtual Connect Box Colon, [1733](#)  
Virtual Connect Box Name, [1735](#)  
Virtual Connect Colon, [1737](#)  
Virtual Connect Depth, [1740](#)  
Virtual Connect Incremental, [1742](#)  
Virtual Connect Name, [1744](#)  
Virtual Connect Report, [1746](#)  
Virtual Connect Report Maximum, [1753](#)  
Virtual Connect Semicolon As Colon, [1754](#)  
Specifications  
    Inductance Wire, [649](#)  
Spice netlist  
    compilation errors and warnings, [2049](#)  
    LVS report, [2049](#)  
Square metric, [511, 579, 666](#)  
Square orthogonal metric, [512, 580, 667](#)  
Stamp operation, [1672](#)  
START keyword, [422](#)  
Statements  
    Inductance Wire, [649](#)  
Strained silicon device calculations, [1833](#)  
Stress effect calculations, [1833](#)  
String constants, [46](#)  
String variables, [1727](#)  
Super-hierarchy, [1746](#)  
SVRF Error specification statement, [1675](#)  
SVRF Error Statement (Table), [2008](#)  
SVRF Message specification statement, [1676](#)  
SVRF Version specification statement, [1677](#)  
SVRF Version Statement Errors (Table), [2009](#)  
Syntax Errors (Table), [2005](#)

— T —

Tables  
    Constraints, [45](#)  
Tables (error and warning)  
    Calibre file i/o errors, [2013](#)  
    Calibre GDSII and OASIS output warnings, [2034](#)  
    Calibre GDSII/OASIS cell and placement exceptions, [2015](#)  
    Calibre GDSII/OASIS input errors, [2017](#)  
    Calibre GDSII/OASIS input warnings, [2027](#)

Calibre nmDRC-H warnings, [2011](#)  
Calibre output warnings, [2036](#)  
Calibre PERC TVF Errors, [2050](#)  
Calibre precision exceptions, [2013](#)  
Cell name/location warnings, [2036](#)  
Connectivity extraction and stamp warnings, [2038](#)  
Connectivity Extraction Errors, [1964](#)  
Constraint Errors, [1938](#)  
Continuous nmDRC errors, [1938](#)  
Density Operation Errors, [1943](#)  
Device Definition Errors, [1945](#)  
Device Layer Operation Errors, [1963](#)  
Device Property Specification Errors, [1959](#)  
DFM Operation Errors, [1948](#)  
DRC Check Map Statement Errors, [1940](#)  
Environment Variable Errors, [1964](#)  
ERC Errors, [1964](#)  
General File I/O Errors, [1992](#)  
General Specification Statement Errors, [2003](#)  
ICrules input warnings, [2042](#)  
ICrules notes, [2042](#)  
Included Rule File Errors, [1967](#)  
Layer and Variable Resolution Errors, [1999](#)  
Layer Definition Errors, [1973](#)  
Layer Statement Errors, [1989](#)  
Layer Type Errors, [2008](#)  
Layout Cell List Statement Errors, [1974](#)  
Layout Cell Match Rule Statement Errors, [1974](#)  
Layout MOS Swappable Properties Errors, [1974](#)  
Litho Operation Errors, [1981](#)  
Location Errors, [1975](#)  
LVS Annotate Devices, [1973](#)  
LVS Cell List statement errors, [1938](#)  
LVS Comparison Messages, [2048](#)  
LVS connectivity extraction errors and warnings, [2044](#)  
LVS Device Type Statement Errors, [1963](#)  
LVS Filter Statement Errors, [1966](#)  
LVS Map Device Statement Errors, [1983](#)  
LVS PERC Load Statement Errors, [1977](#)

- LVS PERC Property Statement Errors, [1979](#), [1981](#)  
LVS Property Initialize Statement Errors, [1975](#)  
LVS Property Map Statement Errors, [1977](#)  
LVS Push Devices Statement Errors, [1975](#)  
LVS Recognize Gates Tolerance Statement Errors, [2001](#)  
LVS Reduce Statement Errors, [1979](#)  
LVS Spice Rename Parameter Statement Errors, [2005](#)  
LVS Split Gate Ratio Statement Errors, [2001](#)  
LVS Stamp errors, [2049](#)  
Macro Errors, [1983](#)  
Maximum results and polygon segmentation warnings, [2056](#)  
Miscellaneous warnings, [2056](#)  
Net Area Ratio Errors, [1985](#)  
Numeric Errors, [1987](#)  
OPC Operation Errors, [1989](#)  
Operation Input Errors, [1967](#)  
Output file open warnings, [2058](#)  
Parasitic Extraction Errors, [1992](#)  
PEX Property Computation Errors, [1997](#)  
PEX Statement Errors, [1996](#)  
Polygon flagging warnings, [2060](#)  
Pre-Processor Directive Errors, [1943](#)  
RDB Conflict Errors, [1999](#)  
Rule Check Statement Errors, [1992](#)  
Rule file errors, [2062](#)  
RuleCheck Group Errors, [1967](#)  
Secondary Keyword Errors, [1971](#)  
SVRF Error Statement, [2008](#)  
SVRF Version Statement, [2009](#)  
Syntax Errors, [2005](#)  
Trace Property Statement Errors, [2006](#)  
Unsupported functionality warnings, [2063](#)  
Variable Statement Errors, [2008](#)  
Tcl Verification Format (see TVF), [1889](#)  
TCOEFF, [1466](#), [1485](#)  
TDDRC operation, [1678](#)  
Temperature coefficients, [1464](#), [1483](#)  
Text  
    hierarchical, [1687](#)  
  
local, [1687](#)  
Text Depth statement, [1687](#)  
Text Layer statement, [1689](#)  
Text Print Maximum statement, [1690](#)  
Text statement, [1685](#)  
Title statement, [1691](#)  
Topex operation, [1692](#)  
Touch Edge operation, [1696](#)  
Touch Inside Edge operation, [1698](#)  
Touch operation, [1693](#)  
Touch Outside Edge operation, [1700](#)  
Trace Property computation language declaration statements, [1877](#)  
description, [1872](#)  
example, [1872](#), [1883](#)  
function reference, [1880](#)  
input pair, [1873](#)  
limitations, [1884](#)  
Trace Property statement, [1702](#)  
Trace Property statement Errors (Table), [2006](#)  
TVF, [1889](#)  
    command reference, [1924](#)  
    command syntax, [1892](#)  
    compile-time, [1891](#)  
        error reporting, [1904](#)  
        file usage, [1903](#)  
    examples, [1932](#)  
    interface for device property calculations, [1827](#)  
    layer definitions, [1896](#)  
    namespace importation, [1921](#)  
    runtime, [1906](#)  
        coding guidelines, [1909](#)  
        commands, [1911](#)  
        error reporting, [1919](#)  
        TVF FUNCTION, [1906](#)  
        TVF layer operation, [1908](#)  
        special character handling, [1901](#)  
TVF layer operation, [1908](#)  
  
— U —  
Unit Capacitance statement, [1718](#)  
Unit Inductance statement, [1720](#)  
Unit Length statement, [1722](#)  
Unit Resistance statement, [1724](#)  
Unit Time statement, [1726](#)

Unsupported functionality errors and warnings  
(Table), [2063](#)

User-defined properties  
device definitions, [1776](#)  
device reduction, [1851](#)  
initialization, [1867](#)  
tracing, [1872](#)

— V —

Variable statement, [46](#), [1727](#)  
Variable Statement Errors (Table), [2008](#)  
Variables, [47](#)  
Vertex operation, [1732](#)  
Via placement, [422](#)  
Via/contact transitions, [417](#)  
VIALAYER, [1264](#)  
Virtual Connect Box Colon statement, [1733](#)  
Virtual Connect Box Name statement, [1735](#)  
Virtual Connect Colon statement, [1737](#)  
Virtual Connect Depth statement, [1740](#)  
Virtual Connect Name statement, [1744](#)  
Virtual Connect Report Maximum statement,  
[1753](#)  
Virtual Connect Report statement, [1746](#)  
Virtual Connect Semicolon As Colon  
statement, [1754](#)

— W —

Warning messages  
Calibre GDSII and OASIS output, [2034](#)  
Calibre GDSII/OASIS input, [2027](#)  
Calibre nmDRC-H, [2011](#)  
Calibre output, [2036](#)  
Cell name/location, [2036](#)  
Connectivity extraction and stamp, [2038](#)  
GDSII/OASIS cell and placement, [2015](#)  
ICrules input, [2042](#)  
LVS compilation, [2049](#)  
LVS connectivity extraction, [2044](#)  
Maximum results, [2056](#)  
Miscellaneous, [2056](#)  
Polygon flagging, [2060](#)  
Polygon segmentation, [2056](#)  
precision, [2013](#)  
Runtime, [2010](#)  
Unsupported functionality, [2063](#)

Well proximity enclosure functions (devices),  
[1815](#), [1833](#)

Width checks (examples), [2073](#)  
WIDTH keyword, [316](#)  
WIDTH1 keyword, [316](#)  
WIDTH2 keyword, [316](#)  
With Edge operation, [1755](#)  
With Neighbor operation, [1757](#)  
With Text operation, [1762](#)  
With Width operation, [1763](#)

— X —

XOR operation, [1768](#)

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

---

## **Third-Party Information**

For third-party information, refer to [\*Third-Party Software for Calibre Products.\*](#)



# End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:  
[www.mentor.com/eula](http://www.mentor.com/eula)

## IMPORTANT INFORMATION

**USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.**

## END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

### 1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
  - 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
  - 1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.
2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of

receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

#### 4. **BETA CODE.**

- 4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

#### 5. **RESTRICTIONS ON USE.**

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

- 5.4. The provisions of this Section 5 shall survive the termination of this Agreement.
6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at <http://supportnet.mentor.com/about/legal/>.
7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.
8. **LIMITED WARRANTY.**
  - 8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."
  - 8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
12. **INFRINGEMENT.**
  - 12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance

to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

- 12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.
- 12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.
- 12.4. **THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.**
13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.
  - 13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.
  - 13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.
14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.
15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.
16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.
18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International

Arbitration Centre (“SIAC”) to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics’ right to bring an action against Customer in the jurisdiction where Customer’s place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
20. **MISCELLANEOUS.** This Agreement contains the parties’ entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066